# CS498: IoT
# Final Project:
# POST (Personal Office Space Tracker)

## May 16, 2021

## Team: PiCognito

**Ujjal Saha – ujjals2**

**Ashish Pradhan – apradh6**

## Table of Contents

## 1. Video Demo

| | |
|---|---|
| **Video URL (Box)** | https://uofi.box.com/s/qgun02m91xhqyn4hzcbkk9cgy4lnflht |
| **Video Duration** | 13:36 mins |

## 2. Code Walkthrough Video Demo

| | |
|---|---|
| **Video URL (Box)** | https://uofi.box.com/s/oaanb7xkrs9jvx6t8qsuarjflxhl8ytg |
| **Video Duration** | 26:12 mins |

## 3. GIT Repo

| | |
|---|---|
| **GIT URL** | https://gitlab.engr.illinois.edu/ujjals2/cs498_iot_spring_2021/-/tree/master/project |
| **Access & Permissions** | Course Instructors and TAs have been provided READ access |

## 4. Motivation

The pandemic has brought about a paradigm shift in almost all aspects of life, be it physical attributes, mental health, social behavior, economic activity and many more. It has had such far-reaching ramifications that no one could have imagined before. Now that almost the entire world's population has faced the consequences, there are some issues which do not get talked of more often as the other topics, mainly because those might seem not that "urgent". One of those is the wholesale shift of the work from home culture for white collar jobs. WFH (Work from Home) has been a boon and a bane, a balancing act which is slowly tipping over to the negative side as time goes on. During this time which has lasted more than a year, we all have faced it, times when we have no idea how long we have worked that day. Earlier, employees would have a fixed time frame where they are supposed to work, a proper 9 to 5 job, where after 5 you can just turn off that phase of your life and move on to the next tasks. However, now as the boundaries have just dissolved due to the home doubling as an office, research[1] has shown that people are spending almost two hours more every day at work.

This has led to a significant detrimental impact on the mental health and the well being of employees. People are suffering more from depression as well as physical problems, due to bad ergonomic practices or otherwise. Normally, in an office space at one's house, there is hardly any interaction which could lead to any sort of correction of issues which could be harmful for the body. But what if there is a device which can keep track of your activities during the time you are at your office desk? A IoT device that could give you a summary of the different movements that has been done throughout the day, tiny yet important metrics which could in the long run make a big difference in the wellbeing of a person? "POST" tries to do exactly that, a virtual assistant that every WFH person might not need, but something that they deserve.

Hence the inspiration for naming the IoT Device as **POST** (**P**ersonal **O**ffice **S**pace **T**racker).

## 5. Objective

The pandemic has brought about POST is designed using already existing hardware that were sourced for the various CS498 IoT Labs that were done previously. Utilizing the concepts and skills using various tools learnt during this course, we reused the PiCar 4WD, the Raspberry Pi4 kit, the PiCamera and a standard USB Microphone to create a virtual assistant which would start the day with some basic announcement, after which it will start tracking key activities performed during work hours which could clarify how healthy of a workstyle the user is living within his WFH hours. These metrics will be sensed, measured, and subsequently stored in a database. The database will then be accessed by a web application and with the help of a dashboard, display the summary of an entire day's work. Not only that, but we have also strived to make it a secure application with basic user recognition techniques to deter unauthorized usage of the virtual assistant. Alerts have also been designed to draw the attention of an authorized user if certain tasks fail or is unfulfilled. All in all, the objective is to become an actual workplace monitoring and assistant performing basic tasks as well as keep an eye on key components which otherwise could easily be ignored by the user while prioritizing other activities.

## 6. POST IoT Device – Hardware and Assembly

Below tables shows the hardware and related parts that were used for the POST IoT Device to be fully functional.

| Device | Image | Description |
|---|---|---|
| **Raspberry Pi 4 Model B 8GB** | | The Raspberry Pi4 Model B was used as the POST IoT device. The Raspberry Pi was de-assembled from the PiCar4wd which was used part of IoT Course Labs. The Raspberry Pi will act as POST IoT device prototype.<br>More Info on Raspberry Pi and its features can be found here: [https://www.raspberrypi.org/](https://www.raspberrypi.org/) |
| **Camera Sensor** | | Raspberry Pi Camera Module V2-8 Megapixel,1080p.<br>The camera will be used for the face and posture detection. |
| **Mic Sensor** | | USB 2.0 Mini Microphone for Raspberry Pi 4 Model B.<br>The mic will be used for the voice detection. |
| **External Speakers** | | Aux port enabled external Speakers connected to Raspberry Pi.<br>The speaker will be used as a output device for POST voice assistant audio output. |
| **Audio Cable** | | 3.5mm Audio Cable.<br>The audio cable connects the speaker with Raspberry Pi |
| **Case for Components assembly** | | Case that holds the components (Raspberry Pi, Camera, mic etc.) together |

Below Fritzing diagram (Figure 1) shows the assembly of the components to form the POST IoT Device:

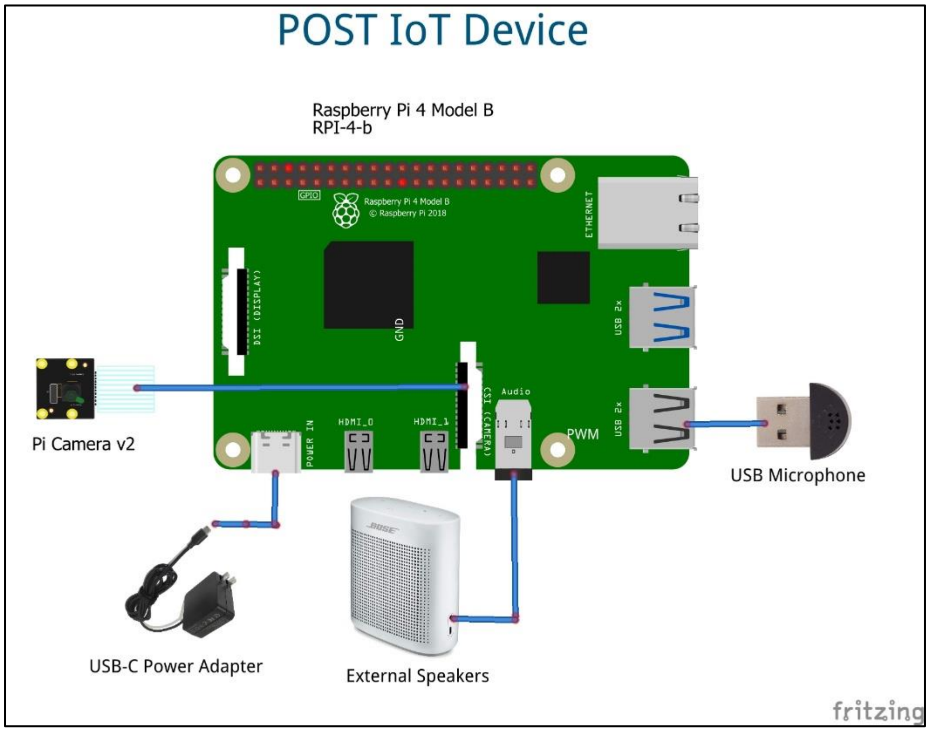


**Figure 1**: Fritzing diagram showing the assembly of components to form the POST IoT Device

After assembly, the POST device would look like below (Figure 2):

(Due to prototype and experiments the camera was stick outside but in real should be part of the casing.)

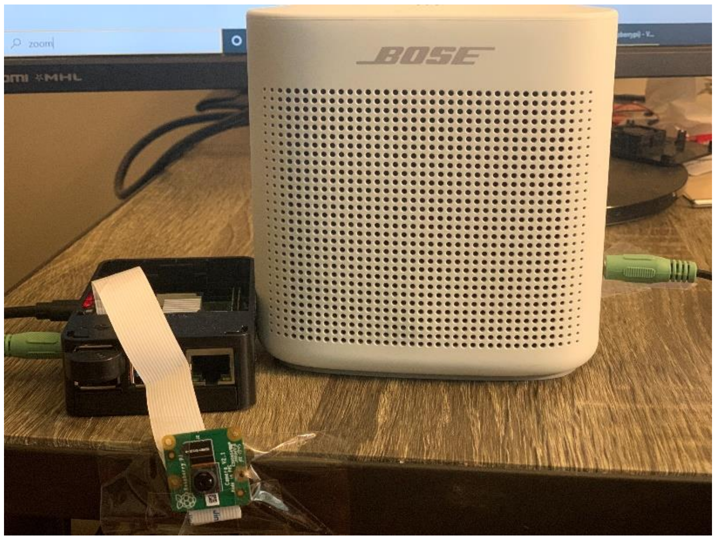


**Figure 2**: POST IoT Device (Prototype and experiment model)

## 7. Project Architecture

POST as a fully functional IoT Device has two aspects in terms of functionality.

a. The POST IoT device that interacts with the user as an monitoring assistant device and also generates data for visualization. The IoT device stores the data in an SQLite3 database and runs http service for sending data.
b. The POST WebApp Dashboard the shows the user analytical data captured by POST in real time interaction using POST's http API.

The below architecture diagram (Figure 3) shows the modules and where they belong in the POST ecosystem:

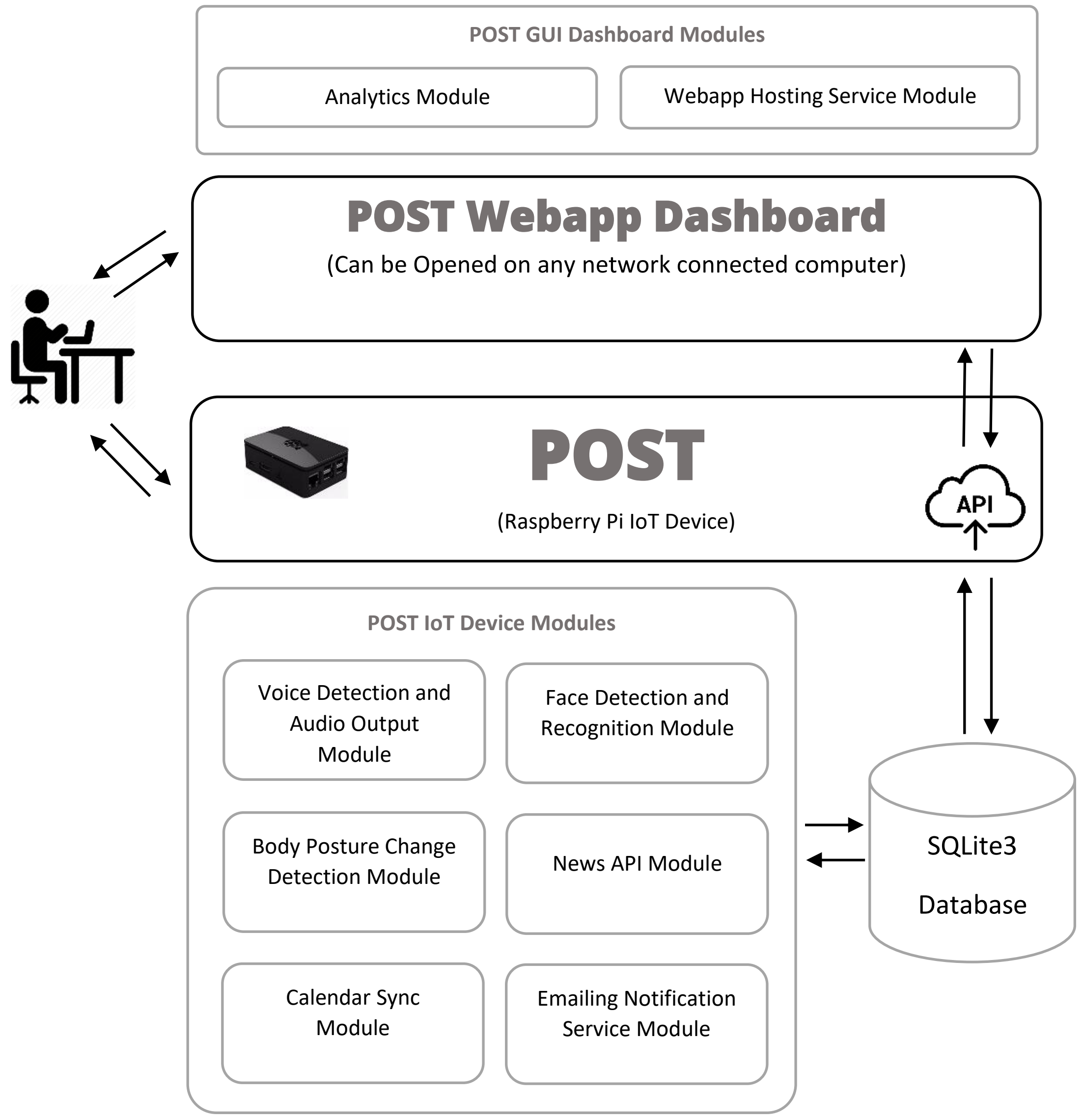

POST GUI Dashboard Modules
Analytics Module
Webapp Hosting Service Module
POST Webapp Dashboard
(Can be Opened on any network connected computer)
POST
(Raspberry Pi IoT Device)
API
POST IoT Device Modules
Voice Detection and Audio Output Module
Face Detection and Recognition Module
Body Posture Change Detection Module
News API Module
Calendar Sync Module
Emailing Notification Service Module
SQLite3 Database

**Figure 3**: Architecture Diagram of the POST IoT Prototype Model

## 8. Module Design and Description

This section explains in detail the design consideration for each module in the above architecture diagram along with roles and responsibilities.

## 8.1. Voice Detection Module

Speech Recognition is an integral part of the entire ecosystem that we are planning to implement using our existing PiCar 4wd hardware. Microphone was the only additional hardware that was required to listen the commands from the user. The microphone that was used for this prototype was manufactured by Sun Founder, the same company that made the PiCar. It is an omni directional noise-canceling mic capable of picking up sound from long distances. There were other higher end microphones available in the market, but this product was chosen as the integration would be seamless with the other hardware, without requiring any additional drivers.

Regarding the software that was used for speech recognition, there are many modules which have been created over the years as this is a booming topic in the field of IoT. There are mainly two types of speech recognition services that are present: Online and Offline. Online services require an internet connection whereas offline ones do not, thus enabling services to run in places where there is no internet or reduced bandwidth. However, there are substantial compromises that need to be made: accuracy as well as speed.

One such offline service is called the Jasper System which requires large disk space to save the system and it takes a lot of effort to pronounce it in such a way that the system can pick up what is being said. There are many online ones which are used to convert voice to text, either from an audio file or in a live environment, with a microphone listening to the commands. The most common ones predominantly use the Google Speech Recognition module or the Google cloud speech API. There are other options as well, the Bing Voice Recognition from Microsoft and Snowboy Hotword Detection being some of the other alternatives.

While exploring all discussed options, it was decided that Google Speech Recognition module would be used to convert the speech to text. Not just that, as the prototype is also created so that it can actively interact with the user using voice, the text spewed out by the speech recognition modules needs to be analyzed and then a relevant response is spoken out by the system. Therefore, there are effectively two distinct parts of this segment.

Here are the steps that were taken to successfully convert speech to text:

1. **Installing libraries:**

   The following libraries are required to run the module:

   a. SpeechRecognition: SpeechRecognition is a very useful library which provides support for several engines and APIs, both online and offline. They are:
      - CMU Sphinx (works offline)
      - Google Speech Recognition
      - Google Cloud Speech API
      - Wit.ai
      - Microsoft Bing Voice Recognition
      - Houndify API
      - IBM Speech to Text
      - Snowboy Hotword Detection (works offline)

The following command was used to install SpeechRecognition:

```
pip3 install speechrecognition
```

b. PyAudio : PyAudio provides python binding for PortAudio, the cross platform audio I/O library. It enables users to easily play or record audio on a variety of platforms like Linux, Windows, MacOS.

The following command was used to install PyAudio:

```
sudo apt-get install python-pyaudio python3-
```

## 2. Get microphone device index

PyAudio enables us to find out the device index which is required so that the SpeechRecognition Module uses that device to listen for commands.

The following snippet of code allows us to find out the list of devices along with the index.

```
import pyaudiop = pyaudio.PyAudio()
info = p.get_host_api_info_by_index(0)
numdevices = info.get('deviceCount')
for i in range(0, numdevices):
    if (p.get_device_info_by_host_api_device_index(0, i).get('maxInputChannels')) > 0:
    print ("Input Device id ", i, " - ",
           p.get_device_info_by_host_api_device_index(0, i).get('name'))
```

After the index was noted(usually its 0 or 1), the final step is to listen for the voice and convert it to text

## 3. Listen and Convert to Text

For this step, the speech_recognition library is used to initialize the Recognizer and the Microphone methods which will listen and subsequently convert the audio to text.

The following script enabled the prototype to achieve this feature:

```
import speech_recognition as srr = sr.Recognizer()


#Include the device index in the line below:
speech = sr.Microphone(device_index=1)
with speech as source:
    audio = r.adjust_for_ambient_noise(source)
    audio = r.listen(source)


try:
    recog = r.recognize_google(audio, language = 'en-US')
    print("You said: " + recog)
except sr.UnknownValueError:
    print("Could you please repeat what you said?")
except sr.RequestError as e:
    print("Unable to access server! Please try again; {0}".format(e))
```

After the prototype finishes listening, the following steps were taken to analyze it and then spoken back to the user.

### 4. Installing pyttsx3

Pyttsx3 is an offline text-to-speech conversion library. To initialize and reference this library, all that is required is to simply call the init() factory function and then its good to go. It includes other widely used TTS engines like sapi5, nsss and espeak. Pyttsx3 installation might require additional libraries to be installed if they are not already present in the system.

The following libraries that need to be upgraded or installed were done so using pip and apt commands:

```
pip install --upgrade wheel
sudo apt update && sudo apt install espeak ffmpeg libespeak1
```

After these were installed successfully, the pyttsx3 was installed:

```
pip install pyttsx3
```

### 5. Convert Text to Speech

Pyttsx3 engine is quite flexible and allows users to modify the properties of the voice generated like volume and the speed in which the voice speaks phrases.

Using the say() factory function, any phrase can be converted to speech. The following script gives a brief example of how pyttsx3 converts text to speech:

```
import pyttsx3
engine = pyttsx3.init()
engine.setProperty('rate', 150)     # Speed percent
engine.setProperty('volume', 0.9)   # Volume 0-1
engine.say("Hello, world!")
engine.runAndWait()
```

## 8.2. Face Detection and Recognition Module

Facial Recognition is incorporated into this prototype as a service which can identify the person who enters the office space. In today’s world, a face can be a quick and secure form of ID which is used  in almost all personal devices like phones, tablets and laptops and also security devices like Ring, in some form or the other. Some are very secure such as that created by Apple which can detect the difference between a photo of the user vs the actual person’s face in front of the sensor. This prototype includes a simplified version of facial recognition using freely available Python libraries and the PiCamera used as the camera hardware.

The following steps were taken to run Facial Recognition successfully in the prototype:

### 1. Install libraries

a. OpenCV (*already installed in our Pi so no need to do anything extra*): It is an open-source software which can process images and videos to identify objects, faces and other abstract objects like handwriting in real time using machine learning.

```
sudo apt install cmake build-essential pkg-config git
sudo apt install libjpeg-dev libtiff-dev libjasper-dev libpng-dev libwebp-dev libopenexr-dev
sudo apt install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev libx264-
dev libdc1394-22-dev libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
sudo apt install libgtk-3-dev libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
sudo apt install libatlas-base-dev liblapacke-dev gfortran
sudo apt install libhdf5-dev libhdf5-103
sudo apt install python3-dev python3-pip python3-numpy
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
mkdir ~/opencv/build
cd ~/opencv/build
```

b. face-recognition: This package is useful for computing the bounding around the faces, compute facial embeddings and compare the faces in the dataset.

   The following command installs it(takes around 20 minutes):

```
pip install face-recognition
```

c. imutils: Imutils provides basic convenience functions which helps in image translation, rotation, resizing etc. It works in tandem with OpenCV and expedites computing image processing using OpenCV.

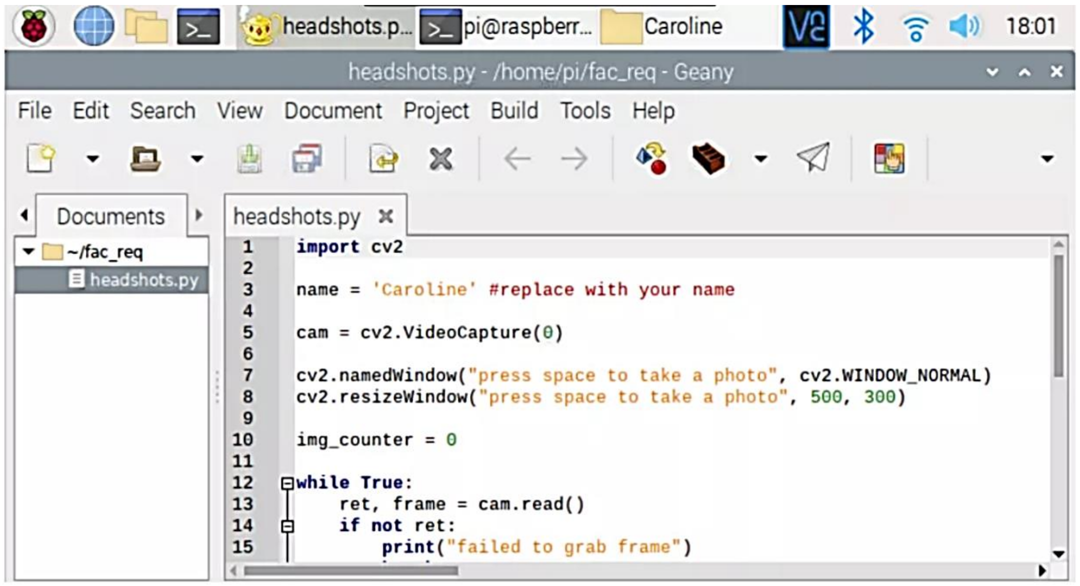
**2. Train the Model**

a. Clone the folder (this will not be included in the final draft)

```
git clone https://github.com/carolinedunn/facial_recognition
```

b. Training the model requires a dataset folder which will include numerous facial images of the person that needs to be identified. It needs to be put as a folder in the dataset folder. (One for each person who needs to be identified).

   Open `headshots.py` script and run it to click multiple photos(different angles of your face) inside the folder. Space bar clicks one photo


headshots.p...
pi@raspberr...
Caroline
18:01
headshots.py - /home/pi/fac_req - Geany
File Edit Search View Document Project Build Tools Help
Documents
~/fac_req
headshots.py
headshots.py

```
import cv2

name = 'Caroline' #replace with your name

cam = cv2.VideoCapture(0)

cv2.namedWindow("press space to take a photo", cv2.WINDOW_NORMAL)
cv2.resizeWindow("press space to take a photo", 500, 300)

img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
```

This can be done for another person and their images too (remember to replace the name to that person)

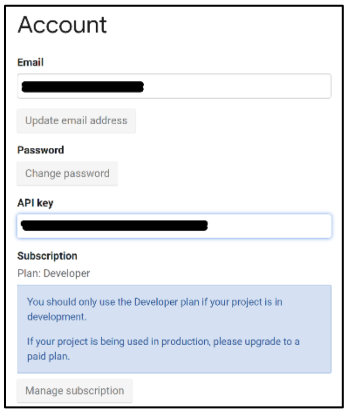
c. After creating the training set, the next step is to navigate to facial_recognition folder(which was cloned). And then run the script train_model.py which will start training the model and then will create a file called `encodings.pickle` which will basically contain the model embeddings.

d. After the script is successfully run, run the script facial_req.py which will open a window with the live feed on and then will be identify the person which has been trained by the model. Facial_recognition library might create a bounding around things other than a face as well which could result in false detections and generate an "unknown" value.

After this facial recognition is good to go. Put the IoTProject_Speech_Face_Recognition.py and postdb.py script in the same facial recognition folder and run the first script. Should work.

## 8.3. News API Module

The news API is a json API belonging to newsapi.org which allows users to get articles and breaking news from reputed sources from all around the world. All that's required is to register on their website, create a Developer account and then generate an API Key through which a python script can search for headlines and curate the output as required by an application.



Account

Email

[redacted]

Update email address

Password

Change password

API key

[redacted]

Subscription

Plan: Developer

You should only use the Developer plan if your project is in development.

If your project is being used in production, please upgrade to a paid plan.

Manage subscription

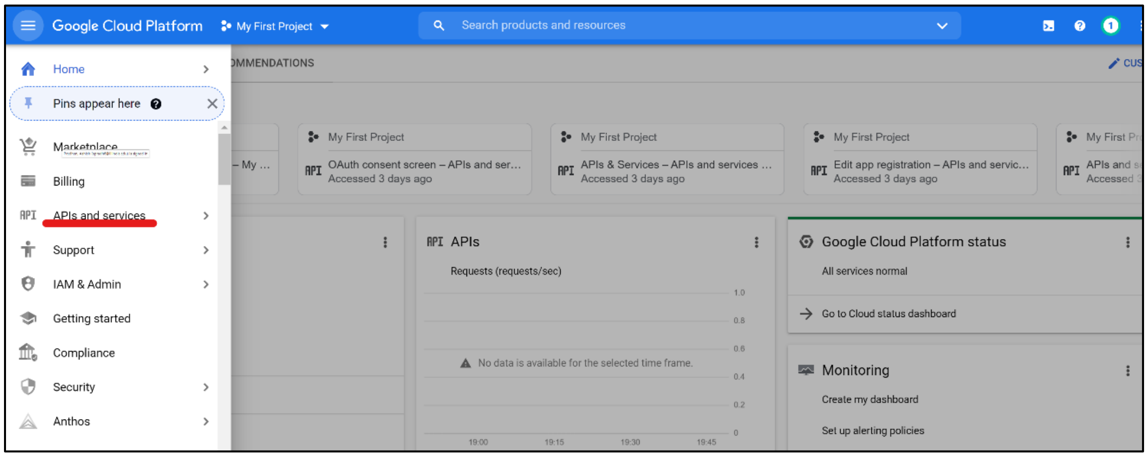
For this assistant we are going to retrieve the top 5 headlines of the day which the Speech Engine will announce. There is a Python client library "newsapi-python" which allows integration of the API into a python application without having to make HTTP requests directly. Here is the code that was used to access the top headlines in US:

```
from newsapi import NewsApiClient
newsapi = NewsApiClient(api_key='XXXXXXXXXXXXXXXXXXXXXXXX')
top_headlines = newsapi.get_top_headlines(country='us')
```

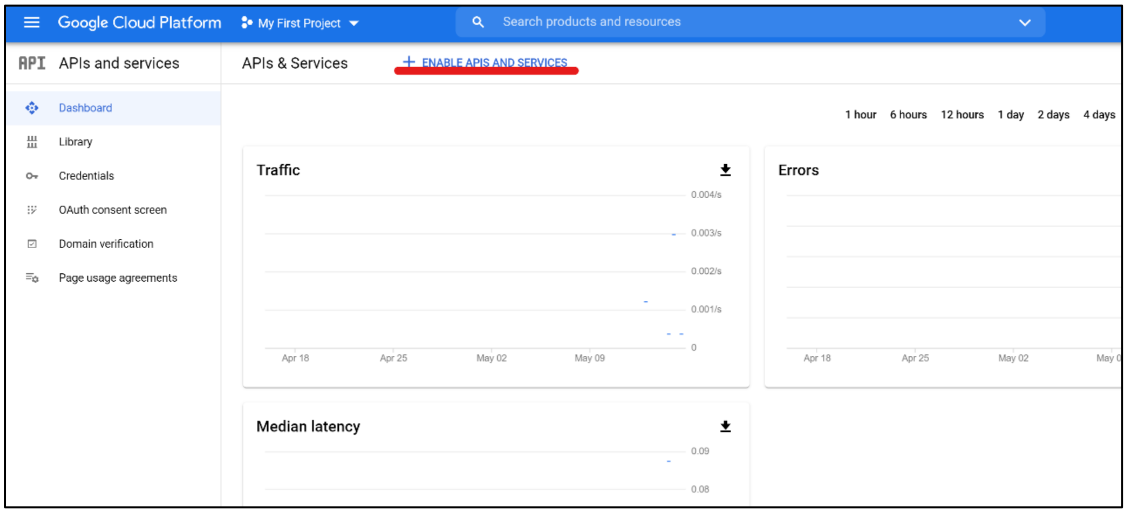
## 8.4. Google Calendar API Module

The Google Calendar API is a fantastic resource which allows a Python application to process requests made to access a Calendar of a particular google account. Successfully incorporating this enabled Post to access the calendar of an account and announce it at the beginning of the day. Here are the steps to install this API:

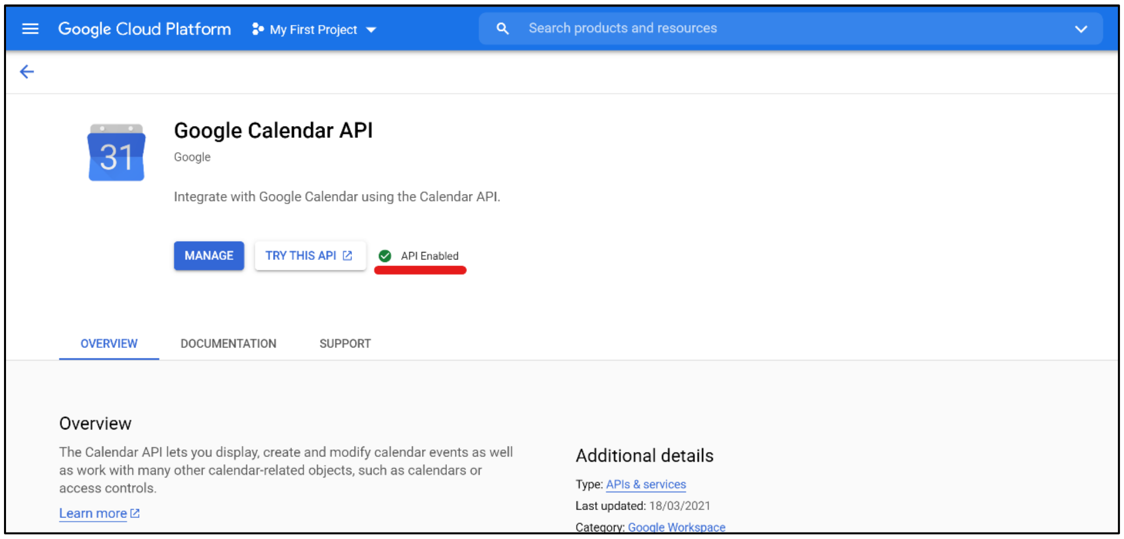
1. **Creating a Google Cloud Platform Account using the account of which the Calendar will be accessed.** Open the Google Cloud Console and create a new Project, filling up the necessary fields. After its created, go to the APIs and Services section and select Dashboard



In the Dashboard, select "Enable APIs and Services"



Search for Calendar API and Enable it.



Google Calendar API

Google

Integrate with Google Calendar using the Calendar API.

Overview

The Calendar API lets you display, create and modify calendar events as well as work with many other calendar-related objects, such as calendars or access controls.

Learn more

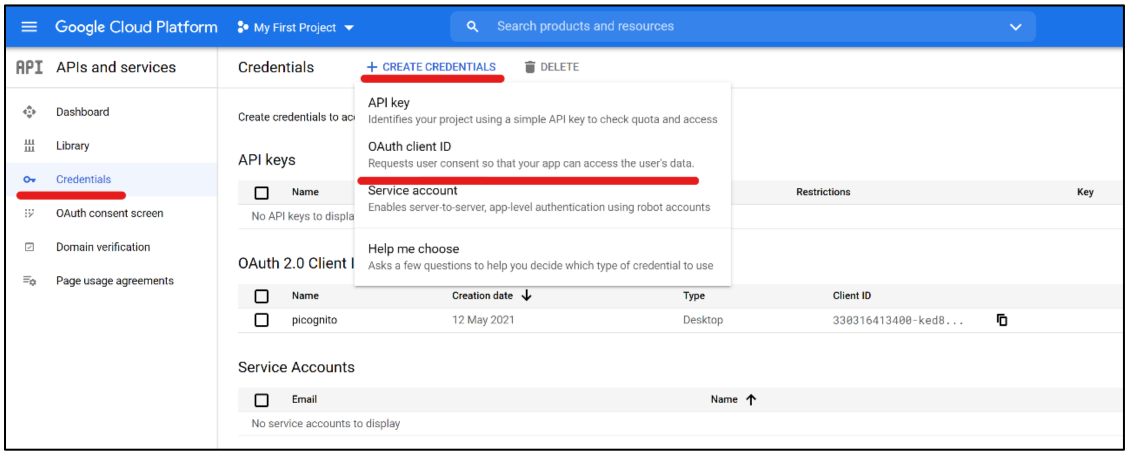
Additional details

Type: APIs & services

Last updated: 18/03/2021

Category: Google Workspace

Navigate back to the Dashboard page and select Credentials. In that page, select Create Credentials and then choose the OAuth client ID



API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so that your app can access the user's data.

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

Create credentials to ac

API keys

| Name | Restrictions | Key |
|---|---|---|
| No API keys to displa | | |

OAuth 2.0 Client I

| Name | Creation date | Type | Client ID |
|---|---|---|---|
| picognito | 12 May 2021 | Desktop | 330316413400-ked8... |

Service Accounts

| Email | Name |
|---|---|
| No service accounts to display | |

Choose the Application Type as Desktop app and then name the ID and hit Create. It will then redirect to the Credentials Page where the newly created ID will be displayed. Select the download option which will download a json file which will be required to access the API from the Python Script.

2. **Install the Google Client Library**

   The following command was run to install the Library in Pi:

```
pip install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib
```

3. **Code Snippet to access Google Calendar Events:**

```
from __future__ import print_function
import datetime
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials

SCOPES = ['https://www.googleapis.com/auth/calendar.readonly']
creds = None
# The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
        if not creds or not creds.valid:
               if creds and creds.expired and creds.refresh_token:
                   creds.refresh(Request())
               else:
                   flow = InstalledAppFlow.from_client_secrets_file(
                       'credentials.json', SCOPES)
                   creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
            service = build('calendar', 'v3', credentials=creds)

    # Call the Calendar API
    now = datetime.datetime.utcnow().isoformat() + 'Z' # 'Z' indicates UTC time
    print('Getting the upcoming 10 events')
    events_result = service.events().list(calendarId='primary', timeMin=now,
                                    maxResults=10, singleEvents=True,
                                    orderBy='startTime').execute()
    events = events_result.get('items', [])
```

## 8.5. Posture Change Detection Module

Pose estimation is a hot topic in the field of Deep Learning where the position and orientation of the object can be determined from images or even a live video feed. This involves pin pointing key parts of that object and track it according as it moves from one coordinate to another in a specific workspace of a 2D view. The primary reason behind adding this into Post was to analyze how often a user gets up or moves around while working a desk job. Staying stagnant sitting on a chair for long hours is considered harmful for the human body and tracking the frequency of the movements can assist in understanding how active a person is throughout his work hours. There are a few pre trained models which have been solely focused on this topic, two of which is PoseNet (a tensorflow lite model https://www.tensorflow.org/lite/examples/pose_estimation/overview) and Multi-Person Pose Estimation by the Perceptual Computing Lab at Carnegie Melon University. We came across a good resource which uses PoseNet on Raspberry Pi and decided to add that feature to Post.

Here are the steps taken:

1. Download and modify the script "run_pose_estimation.py" from https://github.com/ecd1012/rpi_pose_estimation

2. Download the pre trained model from tflite repository using the following command.

```
wget
https://storage.googleapis.com/download.tensorflow.org/models/tflite/posenet_mobilenet_v1
_100_257x257_multi_kpt_stripped.tflite
```

3. PoseNet maps 17 points across the human body when a live video feed is connected to the model.

| Id | Part | Id | Part |
|---|---|---|---|
| **0** | nose | **9** | left Wrist |
| **1** | left Eye | **10** | right Wrist |
| **2** | right Eye | **11** | left Hip |
| **3** | left Ear | **12** | right Hip |
| **4** | right Ear | **13** | left Knee |
| **5** | left Shoulder | **14** | right Knee |
| **6** | right Shoulder | **15** | left Ankle |
| **7** | left Elbow | **16** | right Ankle |
| **8** | right Elbow | | |

For the sake of simplicity, we are only focusing on the change in the position of the nose (part 0). If there is any change in the position of the nose, it is assumed that the user's posture has changed.

## 8.6. Database Engine Module

SQLite3 database has been used for all the storage requirements of POST generated data. The voice data, posture detection data etc are all stored in the `post.db` database within mysqlite3 database. Sqlite3 is inbuilt portable database within python3. To access the sqlite3 from command prompt below command was run on raspberry pi:

```
sudo apt-get install sqlite3
```

Below are the database tables that were created for POST data storage:

```
$ sqlite3 post.db
SQLite version 3.27.2 2019-02-25 16:06:06
sqlite> .schema voice
CREATE TABLE voice (
                          v_id     INTEGER PRIMARY KEY,
                          v_date   DATE NOT NULL DEFAULT (date('now','localtime')),
                          v_time   TEXT NOT NULL DEFAULT (time('now','localtime')),
                          v_speech TEXT NOT NULL
                          );
sqlite> .schema motion
CREATE TABLE motion (
                          m_id   INTEGER PRIMARY KEY,
                          m_date   DATE NOT NULL DEFAULT (date('now','localtime')),
                          m_time   TEXT NOT NULL DEFAULT (time('now','localtime')),
                          m_value TEXT NULL
                          );
sqlite> .schema face
CREATE TABLE face (
                          f_id       INTEGER PRIMARY KEY,
                          f_date     DATE NOT NULL DEFAULT (date('now','localtime')),
                          f_time     TEXT NOT NULL DEFAULT (time('now','localtime')),
                          f_name TEXT NOT NULL
                          );
```

Below are the table description and purposes:

- voice → The voice table will store the timestamp of all voice detection of the user. The data will then be used by dashboard analytics module display voice data visualization
- motion → The motion table will store the timestamp of all posture change detection of the user. The data will then be used by dashboard analytics module display voice data visualization
- face → The face table will store the timestamp of the face detection of the user. This data is used by both POST IoT Device for validating authenticity of the user.

## 8.7. POST Service API Module

The POST Http API service acts as the data provider for the POST WebApp Dashboard. The Http service accepts GET request and based on the requests the api retrieves the data from SQLitee3 database table and sends backs the response in JSON format. The POST Http API is always up and running listening for incoming requests. In this project we are using python3 Flask Package as a lightweight Http API service.

Below are the setup and configurations steps for the Flask API:

1. Install Flask API package
   On Raspberry Pi console below command was run to install the Flask API package
   ```
   pip install flask
   ```

2. Once the flash API has been installed on POST (Raspberry Pi), the GET API methods was written along with the end points:
   ```
   from flask import Flask
   from flask import render_template
   from flask import request, jsonify
   import  json, time
   from postdb import PostDB

   # Simple Web Service using Flask
   app = Flask(__name__)

   # GET  Get Daily login information
   @app.route('/login/',methods = ['GET'])
   def get_login_data():
      pass

   # GET  Get Weekly Voice detection Count
   @app.route('/voice/weekly/',methods = ['GET'])
   def get_weekly_voice_data():
       pass

   # GET Get Daily Voice Detection Count
   @app.route('/voice/daily/',methods = ['GET'])
   def get_daily_voice_data():
       pass

   # GET  Get Weekly Posture Count
   @app.route('/posture/weekly/',methods = ['GET'])
   def get_weekly_posture_data():
       pass

   # GET  Get Daily Posture Count
   @app.route('/posture/daily/',methods = ['GET'])
   def get_daily_data():
       pass

   # Register the flask with the device IP address and Port
   if __name__ == '__main__':
       ip = {'u': '192.168.86.117', 'a':'192.168.10.6'}
       app.run(host=ip.get('u'), port=5000, debug=True)
   ```

3. The POST Http API Service the was run to listen for requests:

```
$ export FLASK_APP=app.py;flask run --host=192.168.86.117
```

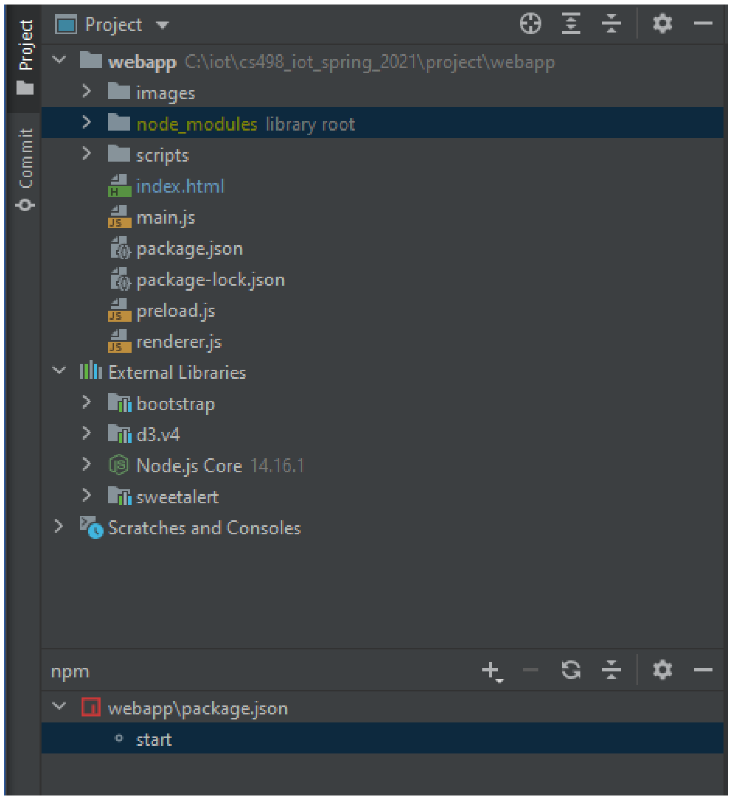
At this point the POST Http Service API is ready to accept request and send json response

## 8.8. POST GUI Dashboard Module

Perhaps one of the most unique features of the POST is to show user data back to the user for self-improvement maintain work life balance while WFH (Working from home). The dashboard shows data visualizations for the voice and posture related monitoring activities what POST was capturing. The dashboard can be run from any machine if the computer is connected to the same network as POST device is. The dashboard gets data by sending http GET request to the POST Http API service. Upon receiving the data the webapp process the data and displays analytical information in form of graphs and charts

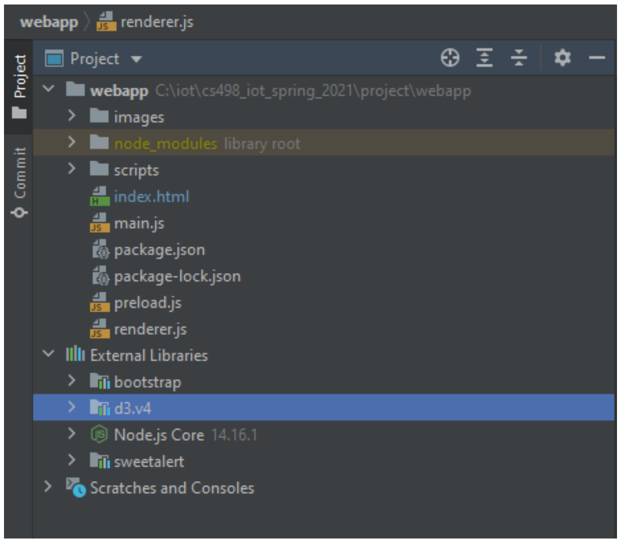
This Webapp dashboard has been developed using node.js 12.14.1, , electron.js 9.4.4 , d3.js v4.0.0 and Chromium 83.0.4103.122. While node.js and electron.js can be setup by following the installation steps but in this project JetBrains WebStorm was used as the development IDE that comes with node.js and electron.js hosting services. WebStorm is a subscription-based IDE provided by JetBrains but comes with 1 month free evaluation. The project was developed using the 1-month free evaluation version of the WebStorm. On detecting the configuration files WebStorm automatically installs the npm service. Below are the main components for the WebApp Dashboard

1. Node.js and electron.js: Below is the snapshot of the Webstorm that shows npm is running based on the project configuration files:

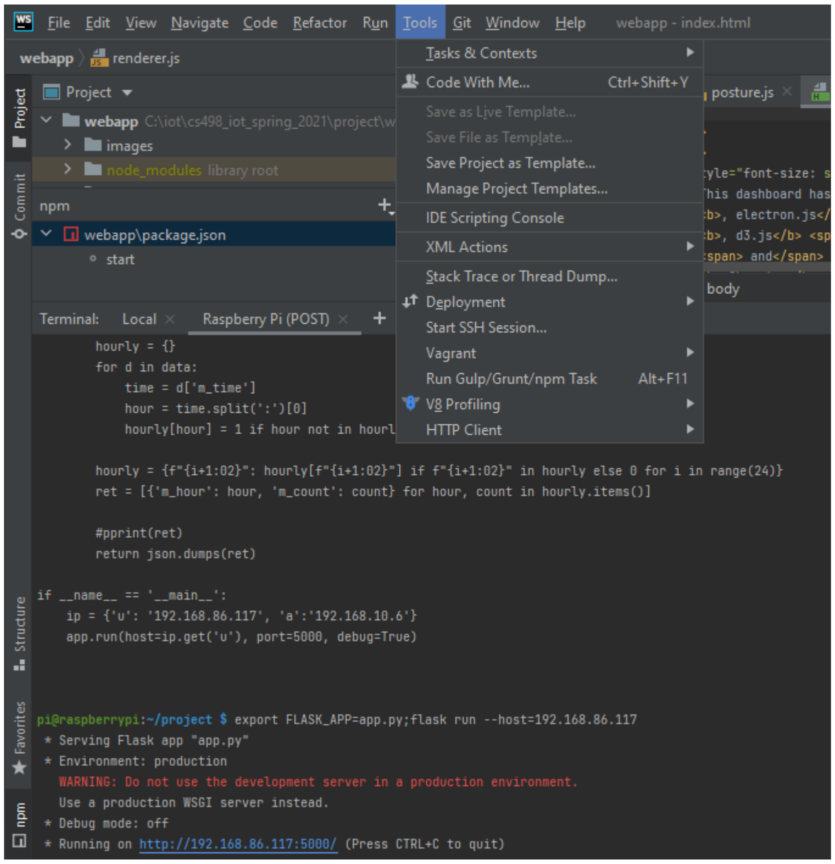


```
Project
webapp C:\iot\cs498_iot_spring_2021\project\webapp
  images
  node_modules library root
  scripts
  index.html
  main.js
  package.json
  package-lock.json
  preload.js
  renderer.js
External Libraries
  bootstrap
  d3.v4
  Node.js Core 14.16.1
  sweetalert
Scratches and Consoles

npm
webapp\package.json
  start
```

2. D3.js: d3.js package was used for all data visualizations on the dashboard. WebStorm automatically downloads the d3.js library once it encounters mention of d3.js in any scripts. Below snahspits shows d3.js has been added as external libraries by WebStorm



3. WebStorm also allows to SSH console to remote access the POST (Raspberry Pi Console) which makes it easier in the development to access remote ssh terminal from the same IDE. Below snapshot shows current the IDE is connected to Raspberry Pi (POST) console via SSH.


File Edit View Navigate Code Refactor Run Tools Git Window Help webapp - index.html
webapp 〉renderer.js
Project
webapp C:\iot\cs498_iot_spring_2021\project\w
images
node_modules library root
npm
webapp\package.json
start
Terminal: Local Raspberry Pi (POST)
Tasks & Contexts
Code With Me... Ctrl+Shift+Y
Save as Live Template...
Save File as Template...
Save Project as Template...
Manage Project Templates...
IDE Scripting Console
XML Actions
Stack Trace or Thread Dump...
Deployment
Start SSH Session...
Vagrant
Run Gulp/Grunt/npm Task Alt+F11
V8 Profiling
HTTP Client
posture.js
body
Project
Commit
Structure
Favorites
npm

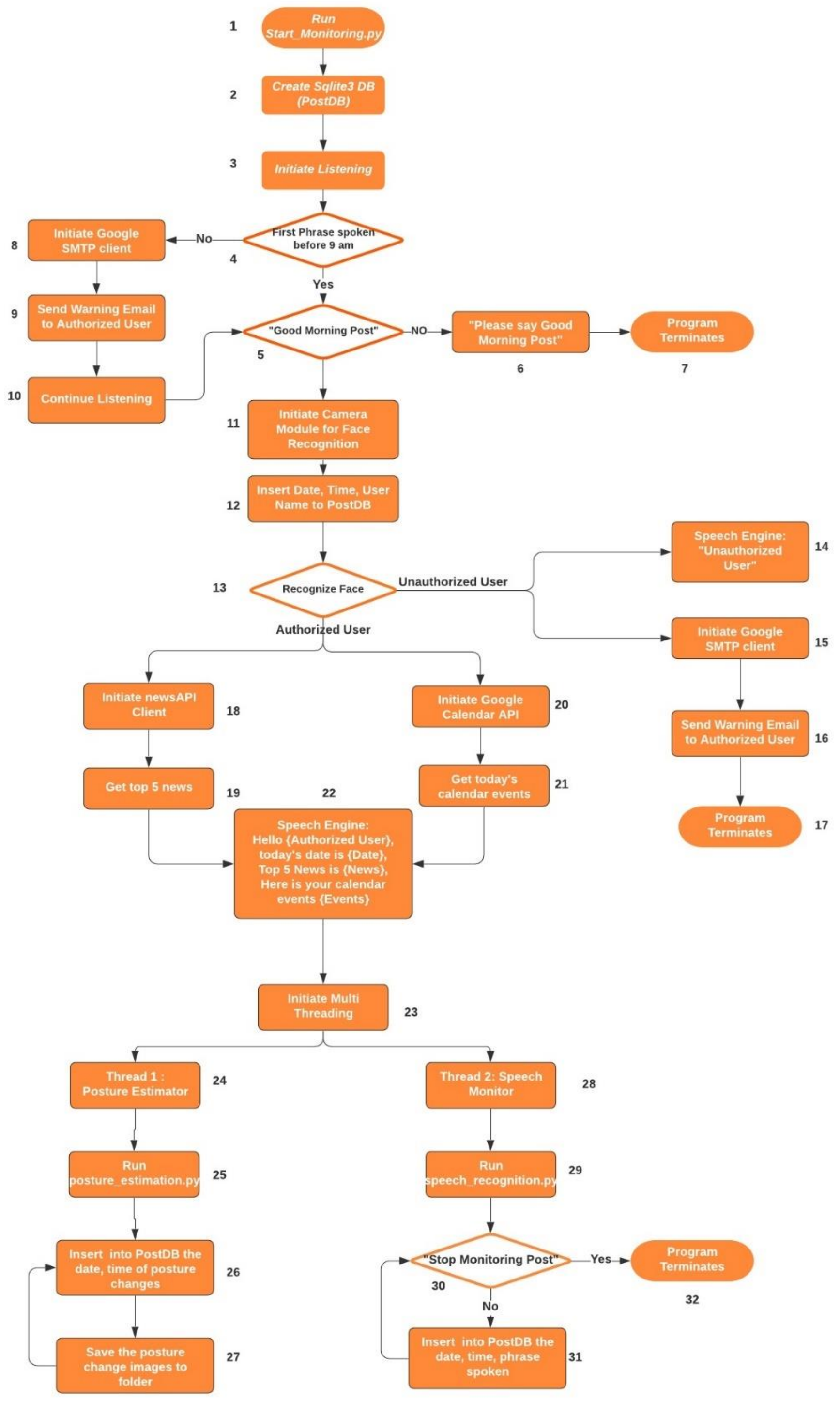
```
        hourly = {}
        for d in data:
            time = d['m_time']
            hour = time.split(':')[0]
            hourly[hour] = 1 if hour not in hourl

        hourly = {f"{i+1:02}": hourly[f"{i+1:02}"] if f"{i+1:02}" in hourly else 0 for i in range(24)}
        ret = [{'m_hour': hour, 'm_count': count} for hour, count in hourly.items()]

        #pprint(ret)
        return json.dumps(ret)

if __name__ == '__main__':
    ip = {'u': '192.168.86.117', 'a':'192.168.10.6'}
    app.run(host=ip.get('u'), port=5000, debug=True)


pi@raspberrypi:~/project $ export FLASK_APP=app.py;flask run --host=192.168.86.117
 * Serving Flask app "app.py"
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://192.168.86.117:5000/ (Press CTRL+C to quit)
```

```
yle="font-size: s
his dashboard has
b>, electron.js</
b>, d3.js</b> <sp
span> and</span>
```

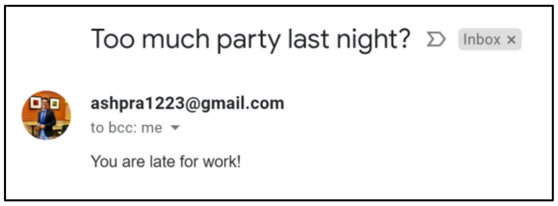
## 9. Functionality and Flowchart

## 9.1. POST IoT Device

Tracking the Processes of Post: The flowchart below shows how the Post processes are connected.


1
Run Start_Monitoring.py
2
Create Sqlite3 DB (PostDB)
3
Initiate Listening
First Phrase spoken before 9 am
4
No
Yes
8
Initiate Google SMTP client
9
Send Warning Email to Authorized User
10
Continue Listening
"Good Morning Post"
5
NO
"Please say Good Morning Post"
6
Program Terminates
7
11
Initiate Camera Module for Face Recognition
12
Insert Date, Time, User Name to PostDB
13
Recognize Face
Unauthorized User
Authorized User
Speech Engine: "Unauthorized User"
14
Initiate Google SMTP client
15
Send Warning Email to Authorized User
16
Program Terminates
17
Initiate newsAPI Client
18
Get top 5 news
19
Initiate Google Calendar API
20
Get today's calendar events
21
22
Speech Engine: Hello {Authorized User}, today's date is {Date}, Top 5 News is {News}, Here is your calendar events {Events}
Initiate Multi Threading
23
Thread 1 : Posture Estimator
24
Run posture_estimation.py
25
Insert into PostDB the date, time of posture changes
26
Save the posture change images to folder
27
Thread 2: Speech Monitor
28
Run speech_recognition.py
29
"Stop Monitoring Post"
30
Yes
Program Terminates
32
No
Insert into PostDB the date, time, phrase spoken
31

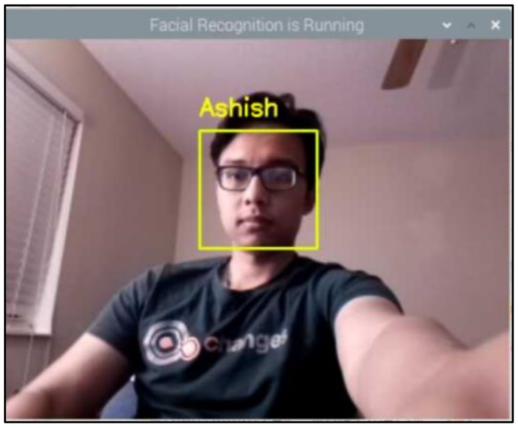
1. Run Start_monitoring.py: This is the starting point of the application. This script downloads all the dependencies as well as create instances of all the processes.
2. Create Sqlite3 DB (PostDB): A sqlite3 database was instantiated with tables: motion, voice, and face.
    a. Motion: Records the date, time of the posture changed.
    b. Voice: Records the date, time and the phrase that was spoken by the user
    c. Face: Records the date, time and the name of the user that is recognized
3. Initiate Listening: The Voice engine as well as the Speech recognition module is initiated. Voice engine imports the package pyttsx3 whereas the speech recognition uses the speech recognition package.
4. First Phrase spoken before 9 am: A script checks whether there has been any entry before or at 9 am on a working day. It is started using a cron job at 9 am every day on a working day
5. “Good Morning Post”: If there is an entry present, it checks whether it is the wake-up phrase “good morning post”.
6. “Please say Good Morning Post”: If it is not “Good Morning Post”, Post will ask to say, “good morning post”.
7. Program Terminates: The application will be terminated.
8. Initiate Google SMTP Client: If there is no entry present before 9 am, it will proceed to send a warning email to the authorized user.
9. Send Warning Email to Authorized User: An email will be sent from the assistant to the email of the user. The library smtplib is used to conduct this process.


Too much party last night?

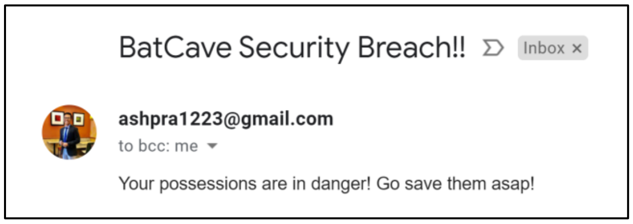
ashpra1223@gmail.com
to bcc: me

You are late for work!

10. Continue Listening: Post will continue listening in the meantime.
11. Initiate Camera Module for Face Recognition: If the phrase is “good morning post”, the camera module will start. PiCamera will start streaming and will recognize the face in front of the camera with the pre trained model.



12. Insert Date, Time, User Name in PostDB: The person identified by the face recognition module will be added into the database along with the date and time when it was done.
13. Recognize Face : Checks if the face is recognized (Here the authorized user is Ashish).
14. Speech Engine "Unauthorized User": If the facial recognition module is unable to recognize the person ("Unknown") then Post will announce the phrase "unknown user".
15. Initiate Google SMTP client: Next, the google email client will be started to get ready to send a warning email to Ashish.
16. Send Warning Email to Authorized User: Ashish will get an alert email from Post indicating there is an unauthorized person in the workspace.


BatCave Security Breach!! Inbox ×

**ashpra1223@gmail.com**
to bcc: me

Your possessions are in danger! Go save them asap!

17. Program Terminate : The application will be terminated.
18. Initiate News API Client: If it is Ashish, news API Client will be initiated using the newsapi module mentioned before.
19. Get Top 5 News: The top 5 headlines and the sources will be fetched using the API.
20. Initiate Google Calendar API: Along with the News API Client, the Google Calendar Client will be initialized
21. Get today's calendar events: The events of today will be fetched
22. Speech Engine…..: After all the information has been retrieved, the speech engine will say the following statement. (This is a sample)

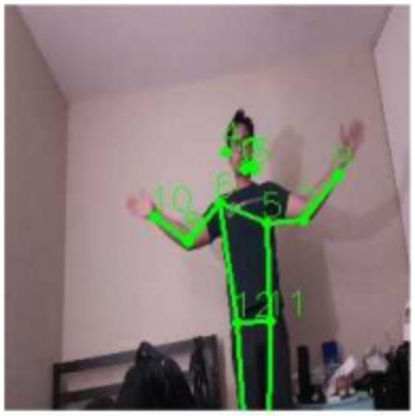
**Sample Voice Output from POST**

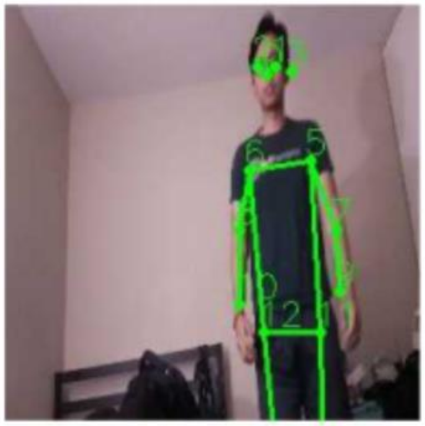
***"Hello Ashish, Today's date is Sunday, May 15th 2021.***

***The top 5 news articles are:***
***China's Tianwen-1 mission nails historic rover landing on Mars - CNET***
***Cops banned from participating in NYC Pride events - New York Post***
***Rombauer Wins the 2021 Preakness, Surging Past Medina Spirit - The New York Times***
***'He's still winning': Kobe Bryant inducted into Basketball Hall of Fame with Tim Duncan, Kevin Garnett – ESPN***

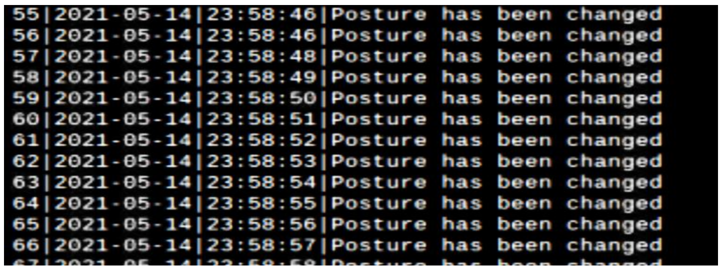
***Here are your Calendar Events:***
***At 11 am, Ashish Pradhan State of IL meeting – 60 Minutes"***

23. Initiate Multi Threading: The monitoring part of the application will now start. There are two major tasks: Speech Monitoring and Posture Change Monitoring
24. Thread 1: Posture Estimator: The posture estimator uses the TensorFlow Lite PosNet to estimate the change in posture of the user.
25. Run posture_estimator.py: PosNet will be run to pinpoint the 17 different coordinates of the human body as mentioned before. If point 0's coordinate moves from one frame to another, that will be defined as a change of posture.



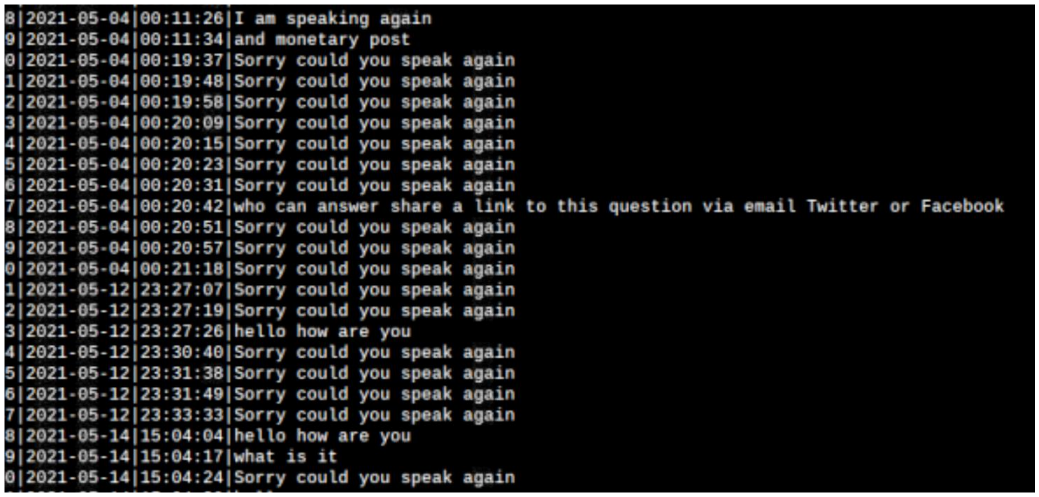


26. Insert into PostDB the date, time of Posture Change: The date and time of the posture change will be inserted into the motion table.



```
55|2021-05-14|23:58:46|Posture has been changed
56|2021-05-14|23:58:46|Posture has been changed
57|2021-05-14|23:58:48|Posture has been changed
58|2021-05-14|23:58:49|Posture has been changed
59|2021-05-14|23:58:50|Posture has been changed
60|2021-05-14|23:58:51|Posture has been changed
61|2021-05-14|23:58:52|Posture has been changed
62|2021-05-14|23:58:53|Posture has been changed
63|2021-05-14|23:58:54|Posture has been changed
64|2021-05-14|23:58:55|Posture has been changed
65|2021-05-14|23:58:56|Posture has been changed
66|2021-05-14|23:58:57|Posture has been changed
```

27. Save the posture change images to folder: The posture change images will be saved in a folder for reference.
28. Thread 2: Speech Monitor: Speech Monitor will be initiated in a separate thread compared to the Pose Estimator so that they can run concurrently.
29. Run speech_recognition.py: This script will have the voice engine and the speech module, like the startup script and will continue listening to the phrases spoken by the user.
30. “Stop Monitoring Post”: Checks if speech module picks up the phrase “Stop Monitoring Post”
31. If it is not so, the module will insert it into the voice table of PostDB along with date and time

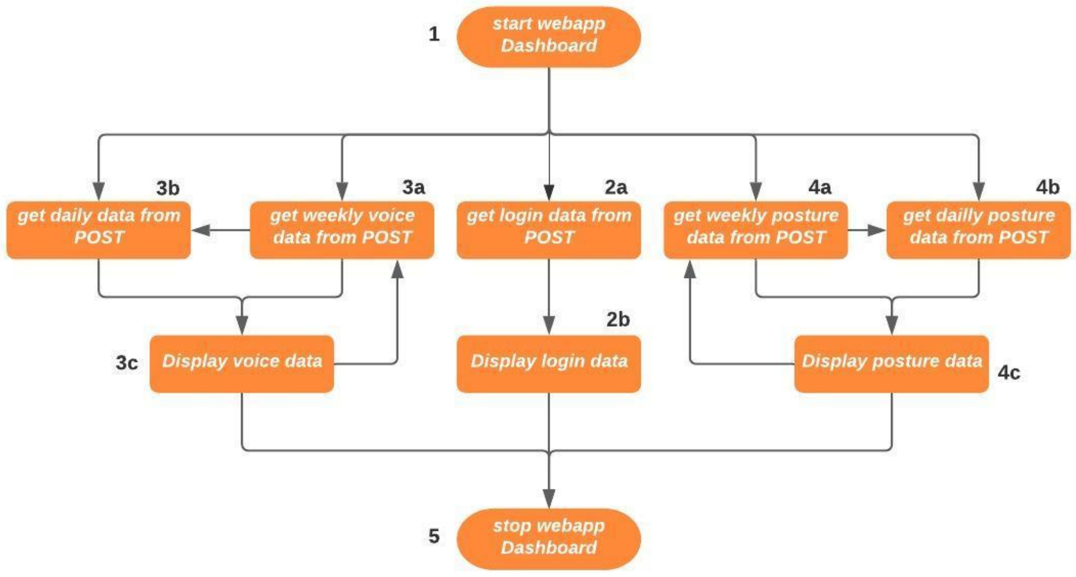


```
8|2021-05-04|00:11:26|I am speaking again
9|2021-05-04|00:11:34|and monetary post
0|2021-05-04|00:19:37|Sorry could you speak again
1|2021-05-04|00:19:48|Sorry could you speak again
2|2021-05-04|00:19:58|Sorry could you speak again
3|2021-05-04|00:20:09|Sorry could you speak again
4|2021-05-04|00:20:15|Sorry could you speak again
5|2021-05-04|00:20:23|Sorry could you speak again
6|2021-05-04|00:20:31|Sorry could you speak again
7|2021-05-04|00:20:42|who can answer share a link to this question via email Twitter or Facebook
8|2021-05-04|00:20:51|Sorry could you speak again
9|2021-05-04|00:20:57|Sorry could you speak again
0|2021-05-04|00:21:18|Sorry could you speak again
1|2021-05-12|23:27:07|Sorry could you speak again
2|2021-05-12|23:27:19|Sorry could you speak again
3|2021-05-12|23:27:26|hello how are you
4|2021-05-12|23:30:40|Sorry could you speak again
5|2021-05-12|23:31:38|Sorry could you speak again
6|2021-05-12|23:31:49|Sorry could you speak again
7|2021-05-12|23:33:33|Sorry could you speak again
8|2021-05-14|15:04:04|hello how are you
9|2021-05-14|15:04:17|what is it
0|2021-05-14|15:04:24|Sorry could you speak again
```

32. If the phrase is “Stop Monitoring Post” the program will terminate.

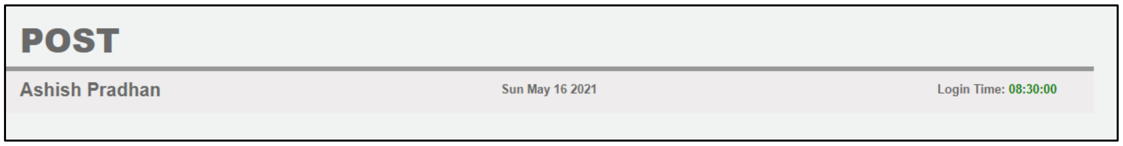
## 9.2. POST Dashboard

Tracking the Processes of Post GUI Dashboard: The flowchart below shows how the Post Dashboard workflow:



1. The user interactive dashboard when started initializes and gather all the request need to visualize the data
2. The get login steps are grouped into below steps:
    a. The "get login data" module calls the POST API to get the user login data.

    b. The login data thus displayed on the screen. Below snapshot shows the user login time displayed in green as the login time is before 9am. If login time is after 9am the time will be displayed in red.



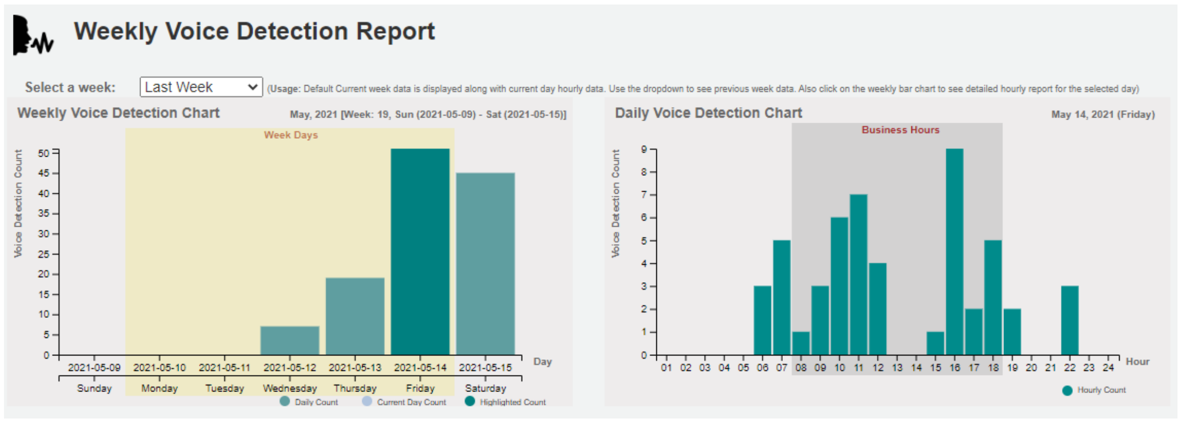
POST

Ashish Pradhan Sun May 16 2021 Login Time: 08:30:00

3. The weekly and daily voice data is grouped into the below steps:
    a. The "get weekly voice data" module calls the POST API to get the voice data for the current week.

    b. The "get daily voice data" module calls the POST API to get the voice data for the current day.

    c. The current week voice data gets displayed. However, if user wants to display any previous week data, user can select from the dropdown to select any specific week. Based on the selected week the dashboard will call the POST service with previous week dates to get corresponding voice data. By default, on the daily chart the current day data will be displayed. But user can click any bar (day) from the weekly chart to see the hourly voice count for that day in the daily chart. The weekly voice detection chart are user interactive charts and based on user inputs updated data visualization will be displayed. Below is the snapshot of the Weekly voice detection Report that displays two charts. The left chart is the weekly voice report, and the right chart is the selected day hourly voice detection.

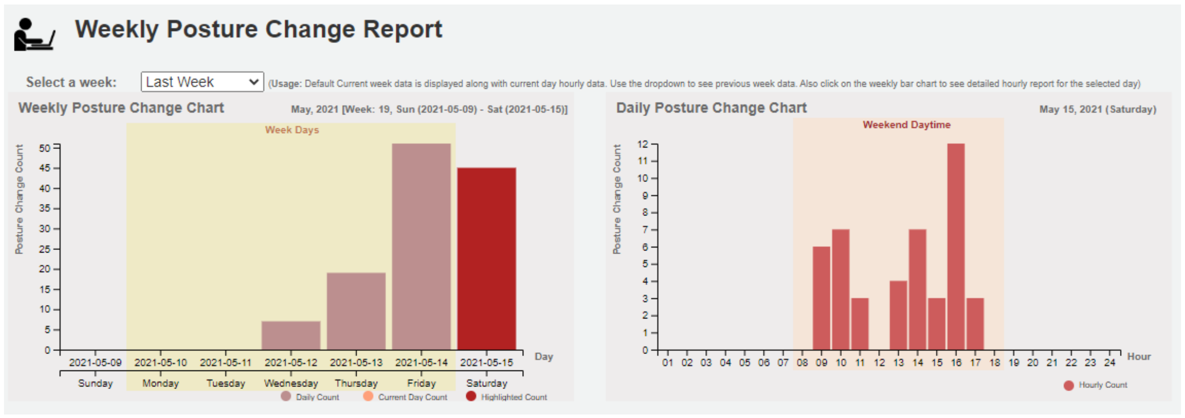


(Usage: Default Current week data is displayed along with current day hourly data. Use the dropdown to see previous week data. Also click on the weekly bar chart to see detailed hourly report for the selected day)

4. The weekly and daily posture data is grouped into the below steps:
   a. The "get weekly posture data" module calls the POST API to get the posture change detection data for the current week.

   b. The "get daily posture data" module calls the POST API to get the posture change detection data for the current day.

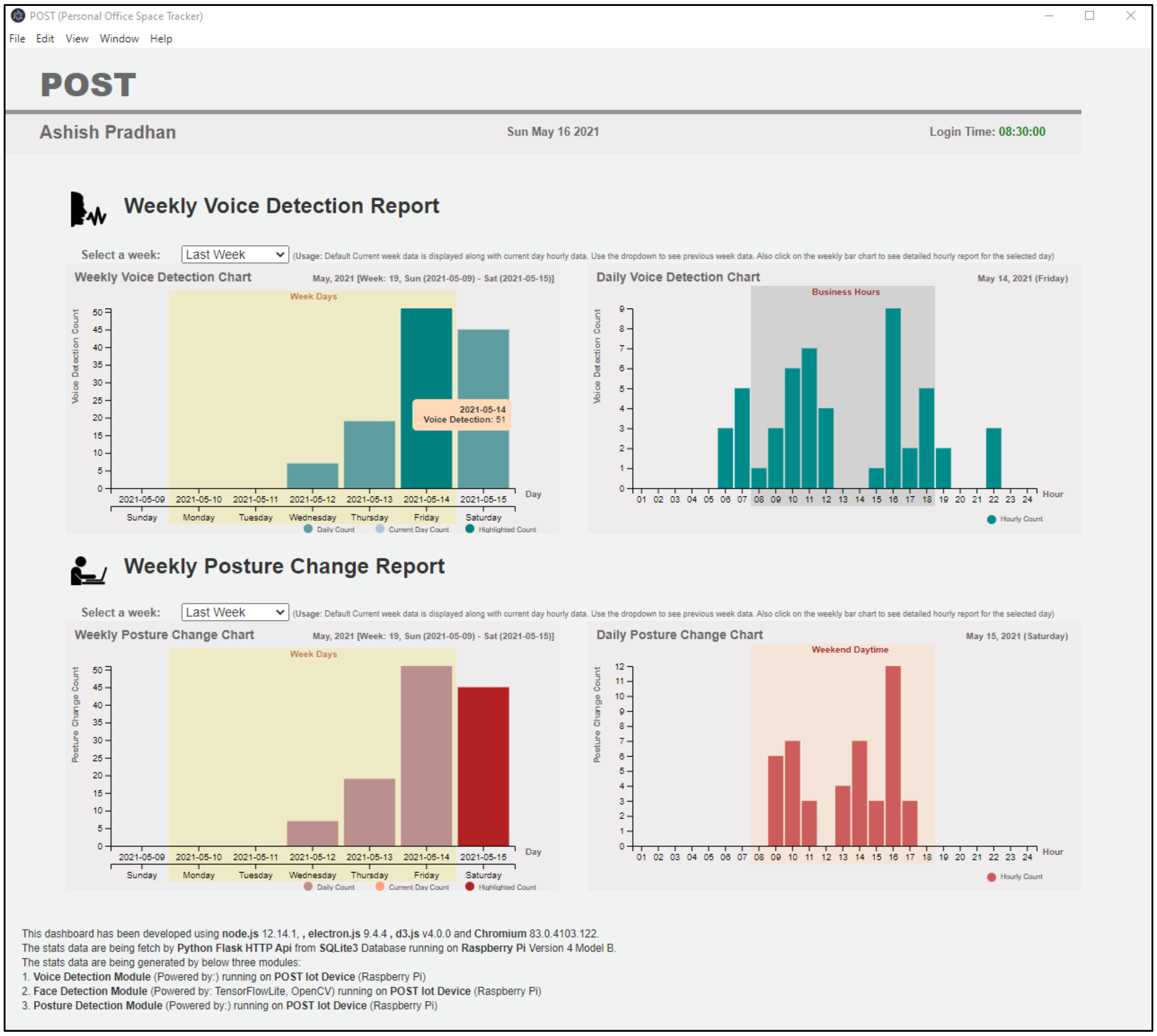
   c. The current week posture change detection data gets displayed. However, if user wants to display any previous week data, user can select from the dropdown to select any specific week. Based on the selected week the dashboard will call the POST service with previous week dates to get corresponding voice data. By default, on the daily chart the current day data will be displayed. But user can click any bar (day) from the weekly chart to see the hourly voice count for that day in the daily chart. The weekly posture change detection chart are user interactive charts and based on user inputs updated data visualization will be displayed. Below is the snapshot of the weekly posture change detection report that displays two charts. The left chart is the weekly posture change detection report, and the right chart is the selected day hourly posture change detection report.



(Usage: Default Current week data is displayed along with current day hourly data. Use the dropdown to see previous week data. Also click on the weekly bar chart to see detailed hourly report for the selected day)

5. At any given instance user can terminate the POST GUI Dashboard and launch again.

Below snapshot shows the complete view of the POST GUI Dashboard. The data visualization are animated displays to keep the user entertaining while displaying the analytical data. The chart visualization was doing using d3.js library. User interactive instruction are provided in the form of Usage for each section. The daily data is displayed for the highlighted day in weekly. The tooltip on the chart provides on-demand additional information when user hovers the mouse pointer over the charts.



POST (Personal Office Space Tracker)
File Edit View Window Help
POST
Ashish Pradhan
Sun May 16 2021
Login Time: 08:30:00
Weekly Voice Detection Report
Select a week: Last Week
Weekly Voice Detection Chart
May, 2021 [Week: 19, Sun (2021-05-09) - Sat (2021-05-15)]
Week Days
Voice Detection Count
2021-05-14
Voice Detection: 51
Day
2021-05-09 2021-05-10 2021-05-11 2021-05-12 2021-05-13 2021-05-14 2021-05-15
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
Daily Count
Current Day Count
Highlighted Count
Daily Voice Detection Chart
May 14, 2021 (Friday)
Business Hours
Hour
Hourly Count
Weekly Posture Change Report
Select a week: Last Week
Weekly Posture Change Chart
May, 2021 [Week: 19, Sun (2021-05-09) - Sat (2021-05-15)]
Posture Change Count
Daily Posture Change Chart
May 15, 2021 (Saturday)
Weekend Daytime

(Usage: Default Current week data is displayed along with current day hourly data. Use the dropdown to see previous week data. Also click on the weekly bar chart to see detailed hourly report for the selected day)

This dashboard has been developed using **node.js** 12.14.1, , **electron.js** 9.4.4 , **d3.js** v4.0.0 and **Chromium** 83.0.4103.122.
The stats data are being fetch by **Python Flask HTTP Api** from **SQLite3** Database running on **Raspberry Pi** Version 4 Model B.
The stats data are being generated by below three modules:
1. **Voice Detection Module** (Powered by:) running on **POST Iot Device** (Raspberry Pi)
2. **Face Detection Module** (Powered by: TensorFlowLite, OpenCV) running on **POST Iot Device** (Raspberry Pi)
3. **Posture Detection Module** (Powered by:) running on **POST Iot Device** (Raspberry Pi)

## 10. Conclusion

POST is a first of its kind device that can assist a person working from home improving the lifestyle showing the daily voice and posture graphs. User can see daily data and also weekly data for comparisons. User also gets notification email if joined late at work and also gets notification if the office place has been visited by an authorized person. Many cool features can be built on top of the device which are discussed in next two sections.

## 11. Challenges Encountered

Below are the major challenges that we encountered during the project and steps were taken to mitigate

1. Lack of experience in web app development. Node.js, electron.js were new areas for exploration. Investing additional hours on some training helped to get used to with it.
2. Posture detection module was very sensitive even a minute movement was being reported. We had to weaken the sensitivity but not capturing all the 17 points data but reduced to only few specific points.
3. Raspberry mic sensor appeared to be not strong, and we had to speak loud for voice detection module to capture the data. We are not sure if it is mic issue or the voice recognition software issue.
4. D3.js programming was a huge challenge as every bit of data visualization is JavaScript based development work. With good training material and online sample references we could expedite the development process.
5. We could find could placement for the camera within the box. We temporarily fixed the issue with a workaround but a raspberry pi case with a camera holder would have been better for development.

## 12. Improvement Areas

Below are some improvement areas and future enhancements that could be done on this project:

1. Voice Recognition library was working but was throwing lot of warnings. Though it did not affect the product from being running but a better voice recognition module would be a nicer fit.
2. Voice recognition sensitivity was not at its best. Any other package could be explored for replacement.
3. The POST WebApp Dashboard can be enhanced to export reports, show side by side data comparison and showing recommendation based on historical data.
4. This device can be enhanced further for endless possibilities of api and services integration and can be used as a personal assistant device (like letting the device control home by tuning lights on/off, dook unlock/lock etc.)
5. AI (machine learning or deep learning) algorithms can be used to make logical prediction of the user workplace trends and alerting for improvements (for e.g., setting a alarm tone at noon to let the user know for a short walk as user historically didn't move much during noon time).

## 13. Team Contributions

Below are the team members roles and responsibilities for the Final Project:

| Name | Contributions |
|---|---|
| **Ujjal Saha**<br><br>NETID: **ujjals2** | 1. Study on the Project Requirements and Proposals<br>2. Components Assembly and Testing<br>3. POC on the Facial Detection, WebApp Dashboard<br>4. Review the POC work of teammate on news api, calendar api, voice detection and posture detection<br>5. Collaborating with the teammate using GIT and developed the modules<br>6. Integration and Testing in a group meeting<br>7. Final Documentation as a group<br>8. Capturing and editing Demo Videos and Final Presentation, Uploads |
| **Ashish Pradhan**<br><br>NETID: **apradh6** | 1. Study on the Project Requirements and Proposals<br>2. Components Assembly and Testing<br>3. POC on news api, calendar api, voice detection and posture detection<br>4. Review the POC work on the Facial Detection, WebApp Dashboard<br>5. Collaborating with the teammate using GIT and developed the modules<br>6. Integration and Testing in a group meeting<br>7. Final Documentation as a group<br>8. Capturing and editing Demo Videos and Final Presentation, Uploads |

# 14. Tools, APIs, Packages and References

Except WebStorm all the packages are either free to use or community edition available under different licenses category.

**WebApp Development**

Reference Article:

https://www.tutorialspoint.com/d3js/index.htm

https://codequs.com/p/B1sJ6MVeI/introduction-to-electron-build-desktop-app-using-node-and-javascript

Packages used:

1. Node.js: Node.js was used for POST GUI Dashboard webapp development
   https://nodejs.org/en/download/
2. electron.js: Electron.js was used for POST GUI Dashboard webapp development (Desktop version)
   https://www.electronjs.org/docs/tutorial/installation
3. d3.js: d3.js was used for data visualizations on the webapp dashboard
   https://github.com/d3/d3/blob/master/API.md

**Python Programming and Database**

Reference Article:

https://www.tutorialspoint.com/d3js/index.htm

https://codequs.com/p/B1sJ6MVeI/introduction-to-electron-build-desktop-app-using-node-and-javascript

Packages used:

1. Python3: Python3 programming language was used for this Project
   https://www.python.org/downloads/
2. Flask: Flask was used as the lightweight Http API service by POST
   https://pypi.org/project/Flask/
3. SQLite3: Sqlite3 was used as the database storage for this project
   https://docs.python.org/3/library/sqlite3.html

**Development Tools**

Packages used:

1. JetBrains WebStorm: Evaluation version was used for the Project WebApp Dashboard Development
   https://confluence.jetbrains.com/display/WI/WebStorm+IDE
2. Postman: Postman was used to test the https api during development
   https://www.postman.com/downloads/
3. Lucidchart: Lucidchart Free version was used for the flowcharts
   https://www.lucidchart.com/

**Voice Recognition:**

Reference Article: https://maker.pro/raspberry-pi/projects/speech-recognition-using-google-speech-api-and-python

Packages used:

1. SpeechRecognition (https://pypi.org/project/SpeechRecognition/)
2. Pyttsx3 (https://pypi.org/project/pyttsx3/)

**Face Recognition:**

Reference Article: https://www.tomshardware.com/how-to/raspberry-pi-facial-recognition

Packages used:

1. OpenCV (https://pypi.org/project/opencv-python/)
2. Face-recognition (https://pypi.org/project/face-recognition/)

**Google CalendarAPI:**

Reference Article: https://developers.google.com/calendar/quickstart/python

Packages used:

1. google-api-python-client (https://pypi.org/project/google-api-python-client/)
2. google-auth-oauthlib (https://pypi.org/project/google-auth-oauthlib/)
3. google-auth-httplib2 (https://pypi.org/project/google-auth-httplib2/)

**News API:**

Reference Article: https://newsapi.org/docs/client-libraries/python

Packages used:

1. Newsapi-python (https://pypi.org/project/newsapi-python/)

**Pose Estimation:**

Reference Video: https://www.youtube.com/watch?v=RUp-K4NEllg

Packages Used:

1. Github Repository (https://github.com/ecd1012/rpi_pose_estimation)
2. PoseNet pretrained model (https://www.tensorflow.org/lite/examples/pose_estimation/overview)