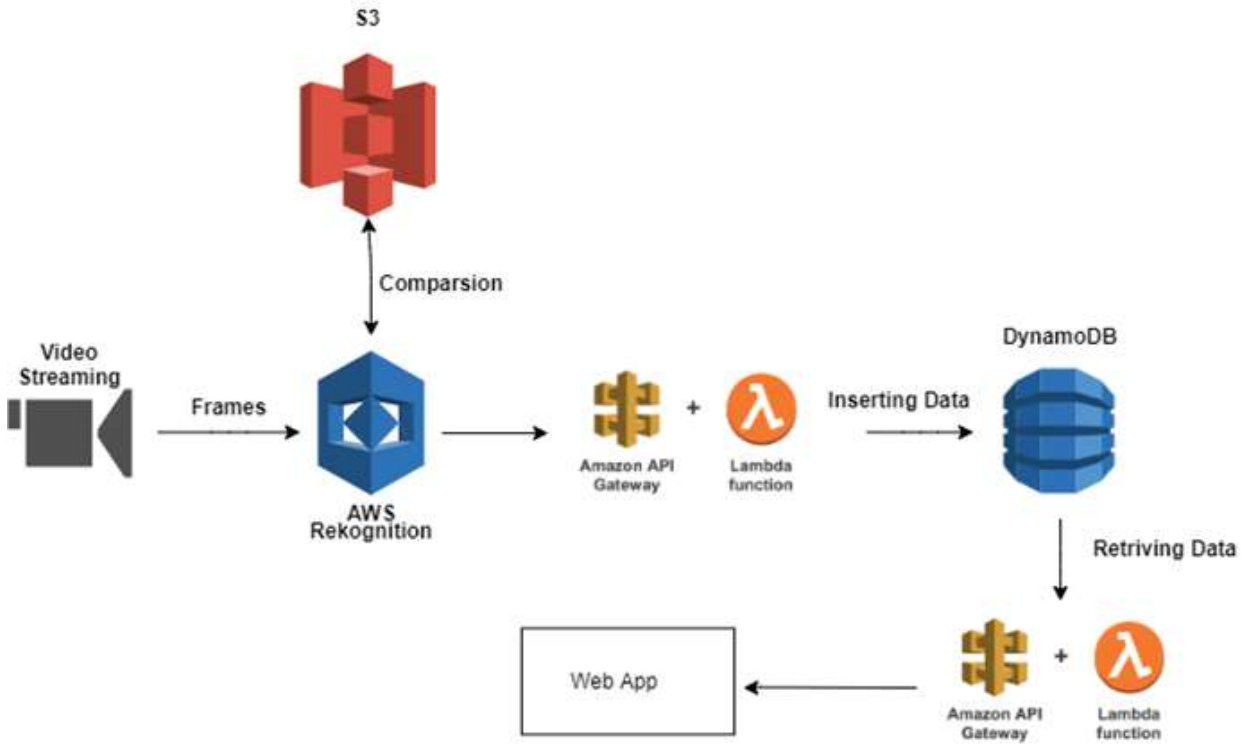
AI-Powered Hourly Attendance Capturing System

Maintaining attendance is very important in all the institutes for checking the attendance percentage of Students. Every institute has its own method in this regard. Some are taking attendance manually using the old paper for every hour and later they will upload every hour data of a class to the server or file-based approach and some have adopted methods of automatic attendance using some biometric techniques. But in these methods students have to wait for a long time in making a queue for every hour. In the Process of making attendance every hour, the students may lose some portion of class every day. So this project focuses on creating an automated system that takes the attendance of students on hourly bases using preinstalled cameras in the classes. We make use of AWS services to marks the attendance, store the attendance in DB



Project Objectives

- AWS Recognition, DynamoDB, lambda functions and API Gateway.
- Hands-on experience on Opencv.
- Know how to create a table in AWS Database
- Work with Http Requests
- Create Rest APIS
- Work with Flask
- Integrate Web App with AWS services

Project Workflow

- Store the Images of Students in S3 Bucket
- create collection ID name for student's faces
- create individual collection IDs
- Capture the image on Hourly basis
- Load the image to Face comparison algorithm (compares the faces in s3 bucket)
- Mark the attendance for compared faces and store in DynamoDB
- Create a rest API using API gateway and lambda function to connect to dynamo DB
- through web app
- Create a web-based dashboard to visualize the attendance

- To accomplish this, complete all the milestones & activities listed below
- Data Collection

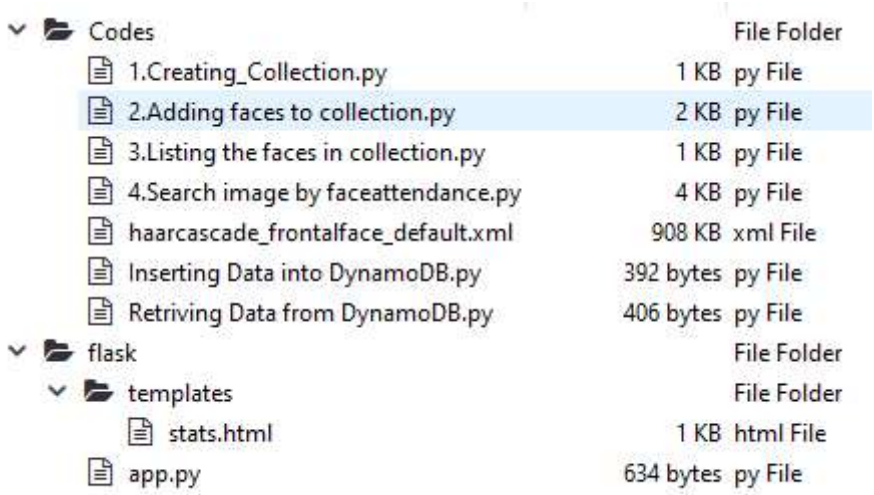
- Collect the images
- Configuring AWS Cloud
- Create an AWS Free Tier Account.
- Create an AWS IAM user.
- Explore an AWS S3 Service.
- Explore an AWS recognition Service.
- Create Collection Ids
- Create face indexes
- Explore an AWS Dynamo DB Service.
- Building a Model
- Write a Python Code for Video Streaming.
- Compare Captured Image with Stored Image.
- Insert the Attendance into DynamoDB through API Gateway.
- Building an Application
- Create a Lamda function to get data from DB.
- Create an HTML Application. (Flask)
- Display the Students attendance in Web App

Prerequisites

- In order to complete this task, you should have an AWS account

- Anaconda Navigator
- TensorFlow
- Keras
- Flask

Project Structure



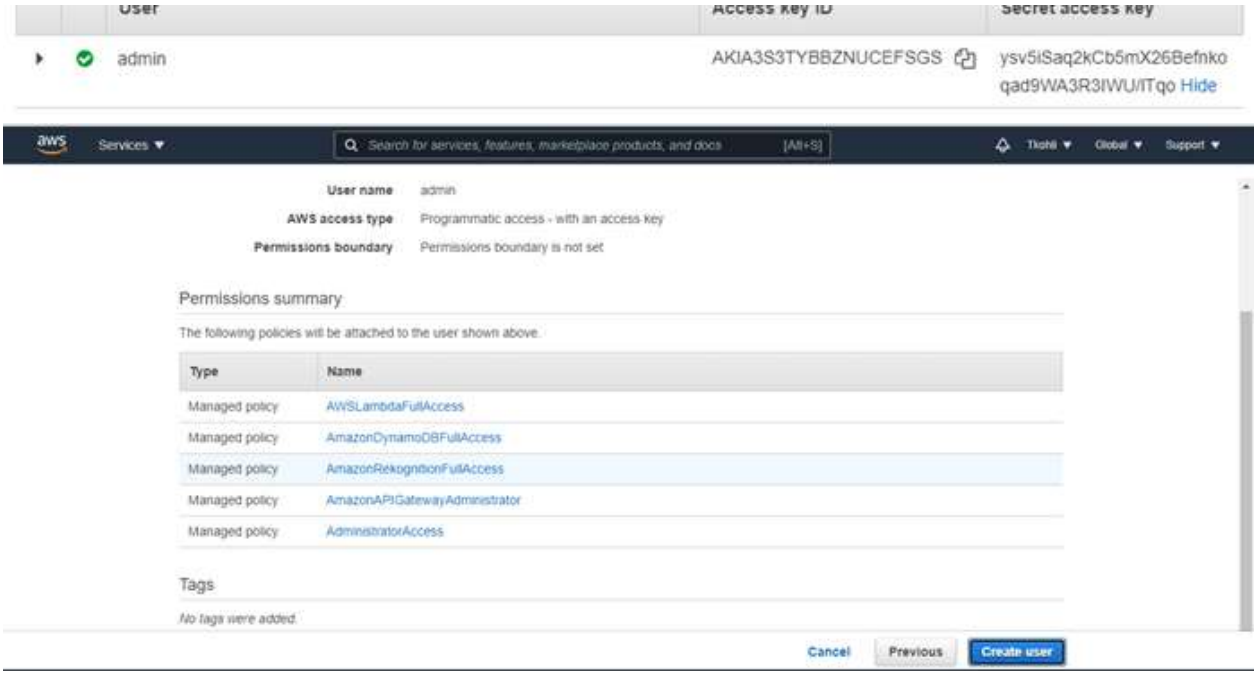
Download the Dataset

For this project, we have collected the images of Sehwag, Ganguly, and Kapil dev. you can download the dataset from the given link

https://github.com/Likhitha5l7/AI-Powered-Hourly-Attendance-System/blob/main/dataset.zip

Configure Aws Account

In order to complete this project first of all you have to configure your Aws Account y creating an IAM user.



Store Images In S3 Bucket

In this activity, let's upload all the images collected to the S3 bucket. These uploaded images will be used to compare the faces from the target image (image capture on hourly base from the class room)

Searching Faces in A Collection

Amazon Rekognition can store information about detected faces in server-side containers known as collections. You can use the facial information that's stored in a collection to search for known faces in images, stored videos, and streaming videos. Amazon Rekognition supports the IndexFaces operation. You can use this operation to detect faces in an image and persist information about facial features that are detected into a collection. This activity is an example of a storage-based API operation because the service persists information on the server.

To store facial information, we must first create (CreateCollection) a face collection in one of the AWS Regions in your account. You specify this face collection when you call the IndexFaces operation. After you create a face collection and store facial feature information for all faces, you can search the collection for face matches. To search for faces in an image, call SearchFacesByImage. To search for faces in a stored video, call StartFaceSearch. To search for faces in a streaming video, call CreateStreamProcessor.

```
In [5]: # Importing of Libraries
        import boto3
        import csv

        # Create client
        client  = boto3.client('rekognition',
                               aws_access_key_id = "AKIA3S3TYBBZMYEBGUNI",
                               aws_secret_access_key = "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",
                                                       region_name = 'us-east-2'
                               )

        def create_collection(collection_id):

            #Create a collection
            print('Creating collection:' + collection_id)

            #Using inbuilt function within rekognition client
            response=client.create_collection(CollectionId=collection_id)

            #Printing the collection details, save the printed output in a text file.
            print('Collection ARN: ' + response['CollectionArn'])
            print('Status code: ' + str(response['StatusCode']))
            print('Done...')

        def main():
            collection_id='tiet-student' #Assign Collection ID Name
            create_collection(collection_id) # Creation of Collection ID

        if __name__ == "__main__":
            main()

Creating collection:tiet-student
Collection ARN: aws:rekognition:us-east-2:796422310002:collection/tiet-student
Status code: 200
Done...
```

Create Facial Features For Collection ID

Here, we are adding the faces from S3 to the created collection Id, in order to generate an array of collections like FaceId, ExternalFaceId etc. The IndexFaces operation enables you to filter the faces that are indexed from an image. With IndexFaces you can specify a maximum number of faces to index, or you can choose to only index faces detected with a high quality.

You can specify the maximum number of faces that are indexed by IndexFaces by using the MaxFaces input parameter. This is useful when you want to index the largest faces in an image and don't want to index smaller faces, such as faces of people standing in the background.

IndexFaces filters faces for the following reasons:

- The face is too small compared to the image dimensions.
- The face is too blurry.
- The image is too dark.
- The face has an extreme pose.
- The face doesn’t have enough detail to be suitable for face search.

To see whole code refer to AI notebook.pdf in github

```
Results for sehwag.jpg
Faces indexed:
  Face ID: f770d2a5-931a-4f2a-9dbe-75b4b1c9636f
  External Id:sehwag.jpg
  Location: {'Width': 0.3840073347091675, 'Height': 0.4242989122867584, 'Left': 0.44988515973091125, 'Top': 0.1449604630470276}
Faces not indexed:
Faces indexed count: 1
Results for Ganguly.jpg
Faces indexed:
  Face ID: be599684-9331-4347-92ad-61340717842f
  External Id:Ganguly.jpg
  Location: {'Width': 0.2629672586917877, 'Height': 0.5258278846740723, 'Left': 0.3263614773750305, 'Top': 0.18856088817119598}
Faces not indexed:
Faces indexed count: 1
Results for kapildev.jpg
Faces indexed:
  Face ID: 0d684cc8-956b-4051-86cd-39d049161ded
  External Id:kapildev.jpg
  Location: {'Width': 0.19932161271572113, 'Height': 0.35589325428009033, 'Left': 0.40642550587654114, 'Top': 0.146672606468200
68}
Faces not indexed:
Faces indexed count: 1
```

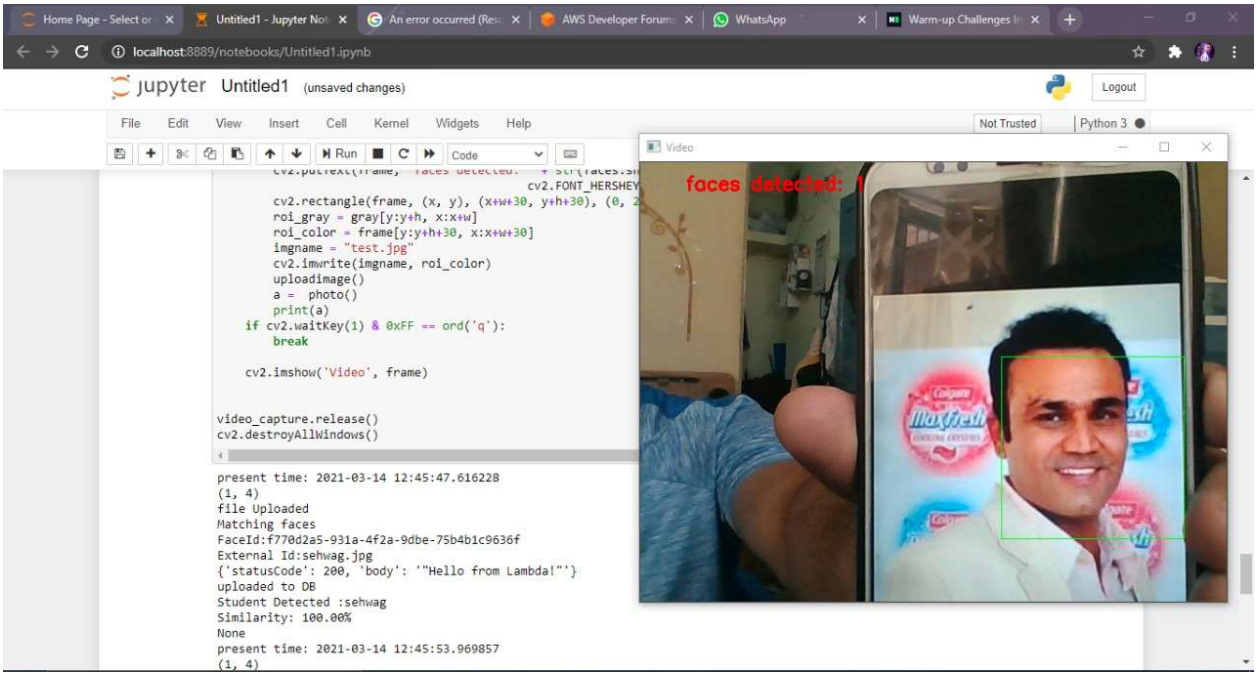
Listing Faces from Collection

The listing faces from the collection describe the face properties such as the bounding box, face ID, image ID of the input image, and external image ID that you assigned

```
Faces in collection : tiet-student
Face Id     : 0d684cc8-956b-4051-86cd-39d049161ded
External Id : kapildev.jpg
Face Id     : be599684-9331-4347-92ad-61340717842f
External Id : Ganguly.jpg
Face Id     : f770d2a5-931a-4f2a-9dbe-75b4b1c9636f
External Id : sehwag.jpg
faces count: 3
```

Comparing Captured Image With Stored Image

Now that We have stored the facial information in the collections, lets use this information to compare the image coming from a video frame or a new picture



Insert Attendance Into DynamoDB Through API

Gateway

As from the above milestone, you can see our face recognition service is perfectly Working fine with our collected dataset, it time for us to develop a script for an hourly attendance marking system. this includes the following procedure

1. capture the image on hourly bases

2. upload the capture face image to s3

4. Compare the captured faces with s3 and fetch the matched details

5. Check the time

6.And store the attendance of each person in Dynamo DB

In order to store attendance. you should first create a Dynamo a table which contains the class period and the name of the student who is present.

Now to upload data to Cloudant we should create an API gateway http url which we will be using in our python script

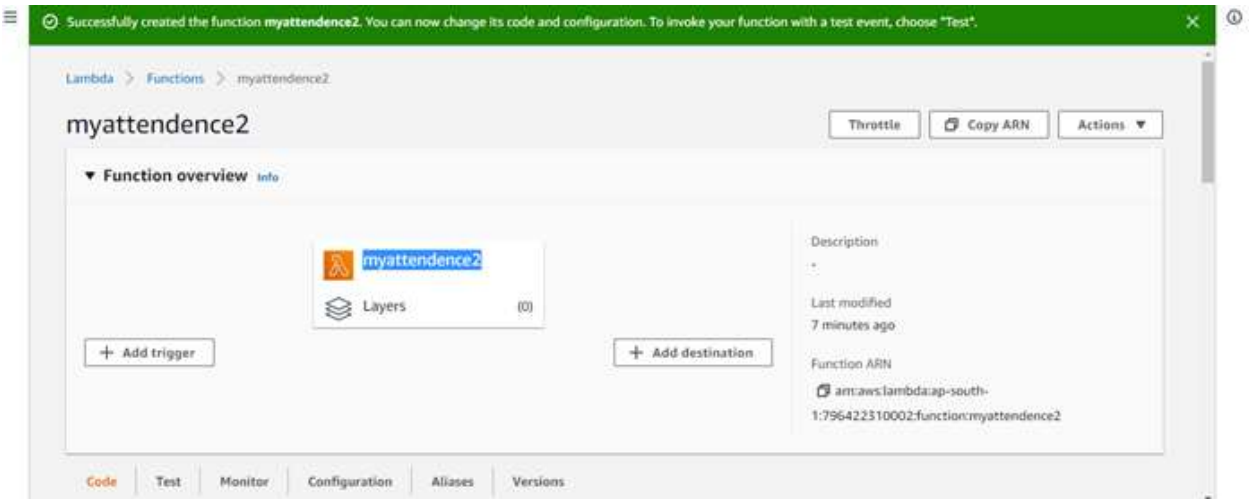
Create Dynamodb Table

Search for the DynamoDB Service from the services & click on it, so that you will be directed to DynamoDB Dashboard as below.

Create Lamda funtion

In order to generate an API (Http url), we should first create a Lambda function. you Search for lambda function service under the compute services of AWS Console Service.

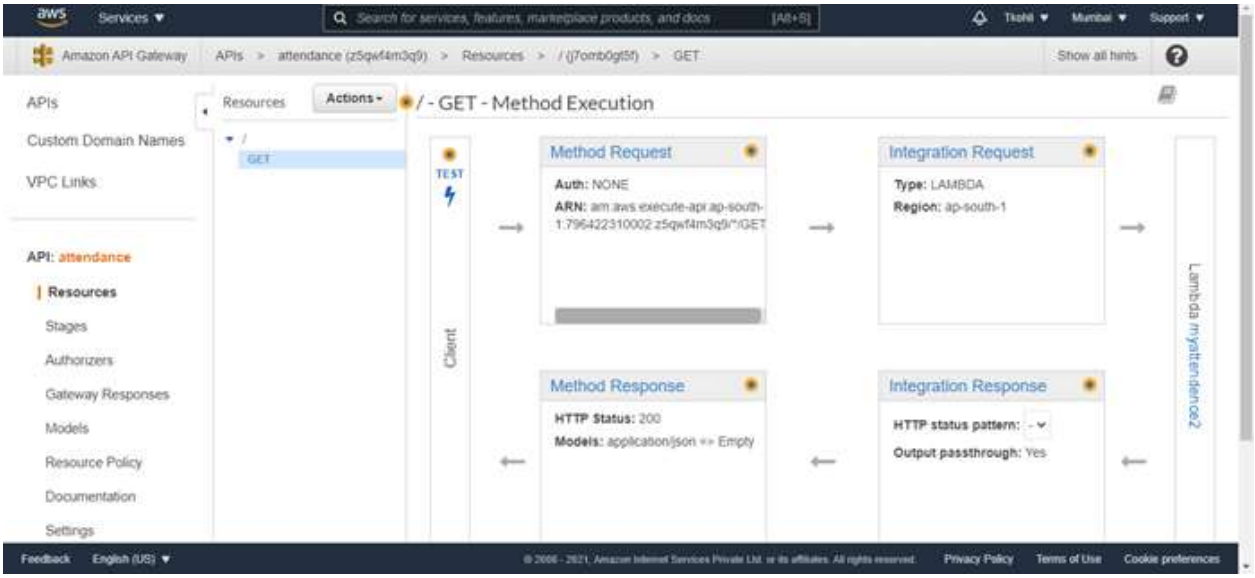
Once the lambda function is opened, click on the **Create function**



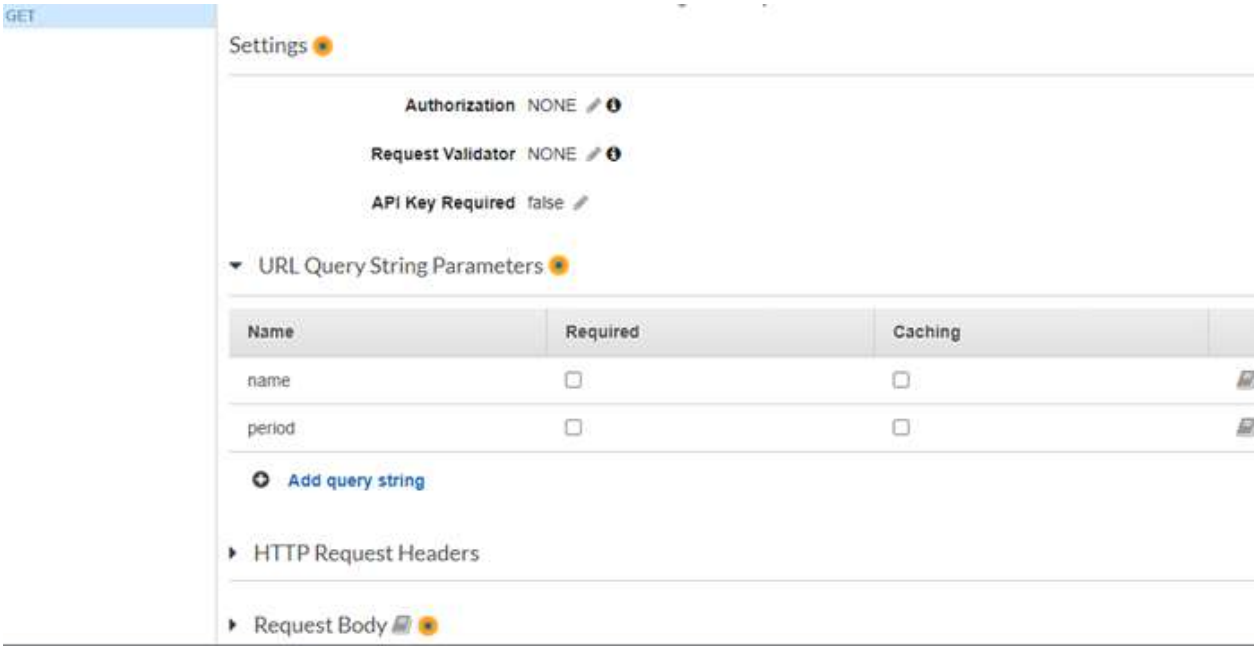
Create an API

Now, search for the API Gateway service under the Networking & Content Delivery service of AWS Console.

Once, API Gateway service is opened, click on Create API



add following params



Deploy the API and then

timestamp	name	period
2020-10-27 07:15:18.442353	Ganguly	period1
2020-10-27 07:15:50.666018	Ganguly	period2
2020-10-27 08:41:29.546952	Ganguly	period3
2020-10-27 08:43:22.639332	sehwag	period1
2020-10-27 17:40:05.656592	Ganguly	period3

Building An Application (Flask)

In this milestone, we are building a web application for the user interface to check the attendance of each student, the total number of detections.

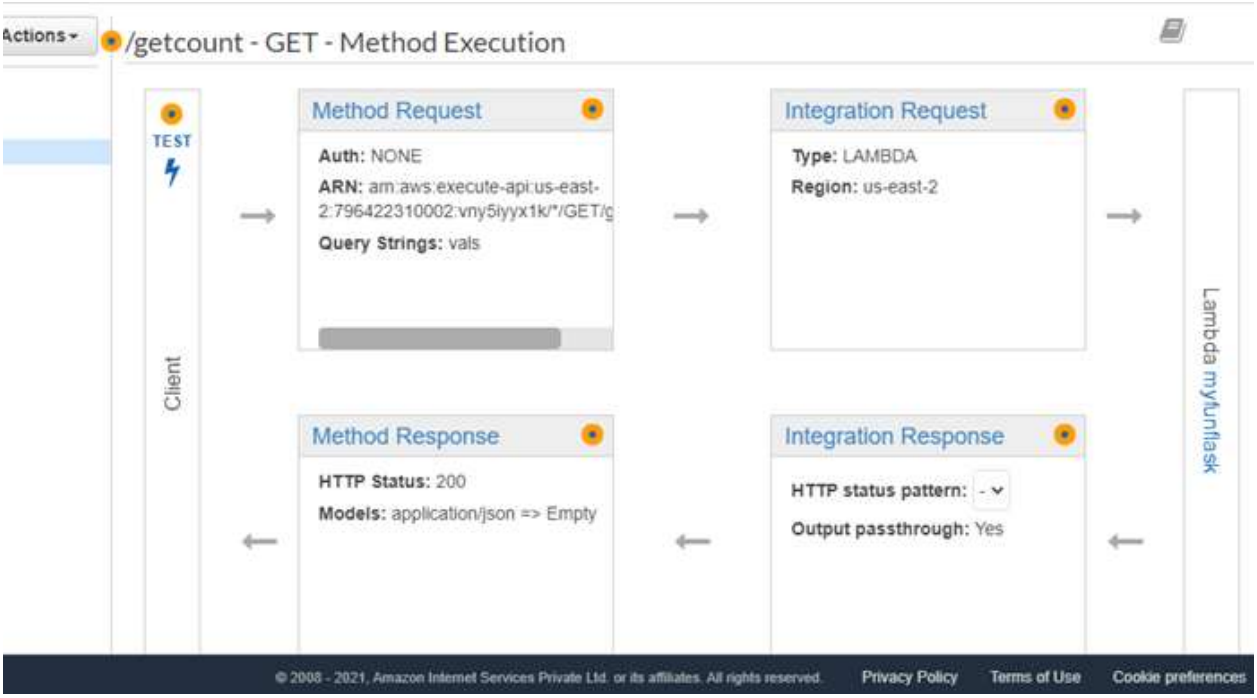
Follow & complete the below activities, in order to accomplish this milestone.

- Create a Lambda Function to retrieve data
- Create API using API gateway
- Showcase Attendance on Flask User Interface

In lamda funtion

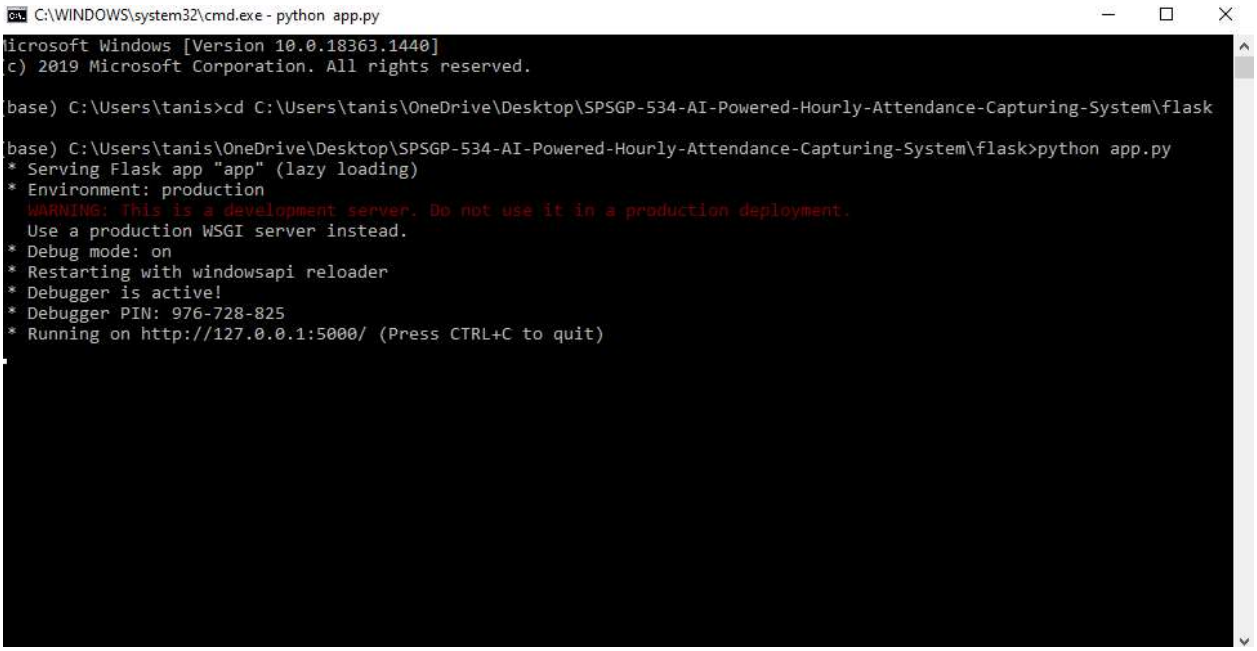
```
import boto3
from boto3.dynamodb.conditions import Key
dynamoDB = boto3.resource('dynamodb')
table = dynamoDB.Table('attendance2')

def lambda_handler(event,context):
    grps = ["Ganguly","sehwag","kapildev"]
    vals = []
    for i in grps:
        response = table.scan(FilterExpression=Key('name').eq(i))
        print(response)
        vals.append(len(response['Items']))
    return vals
```

Then create an API with parameters "vals"



The we write code for HTML and app in flask

then we run the app



And we get the output when showing shewag's image in webcam as