"Face Recognition-Based Attendance Management System"

Introduction

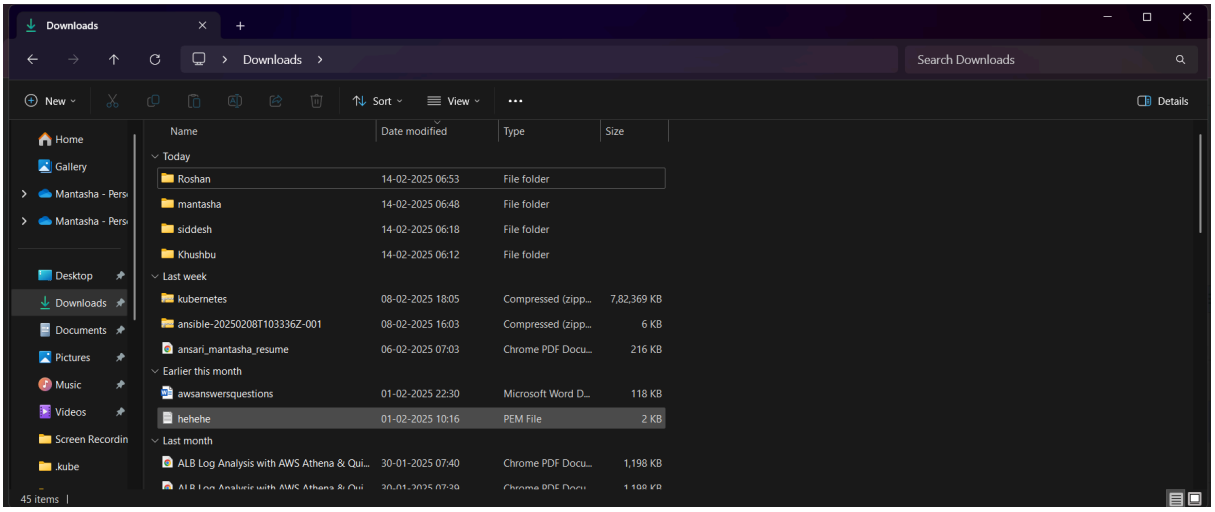
Create an intelligent device that seamlessly interfaces with a camera to capture images at hourly intervals. These images are then transmitted to a trained machine learning model, which harnesses the power of AWS Rekognition Service to accurately identify the faces of students. The recognized images are subsequently stored in an Amazon S3 (Simple Storage Service) repository for secure data retention. Moreover, the model efficiently updates attendance records in a database, automating the process and minimizing manual effort. To provide a comprehensive view of student attendance data, a user-friendly web-based dashboard is developed, offering an intuitive interface for visualizing and managing attendance information.

BLOCK DIAGRAM

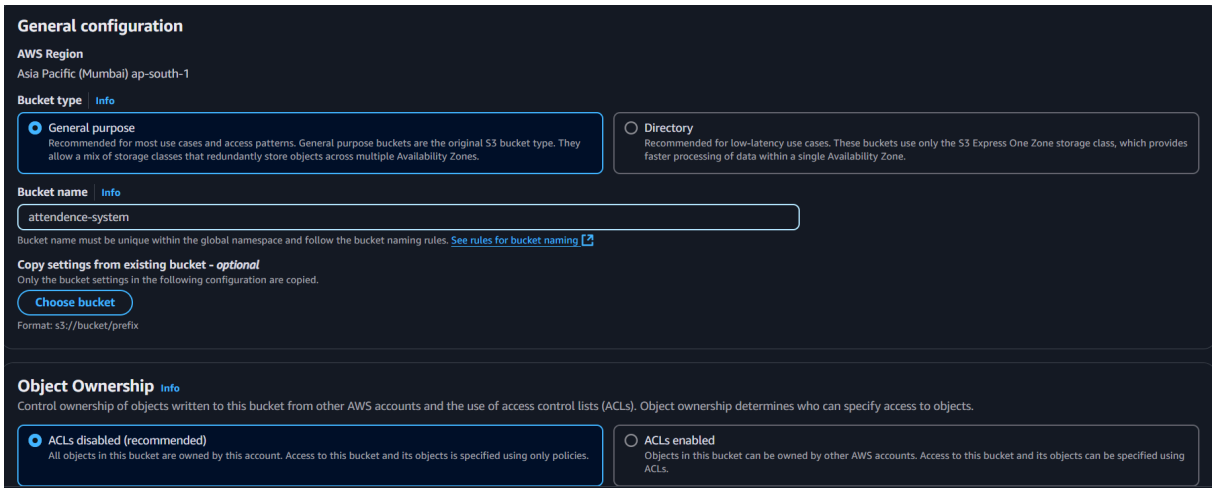
IMPLEMENTATION

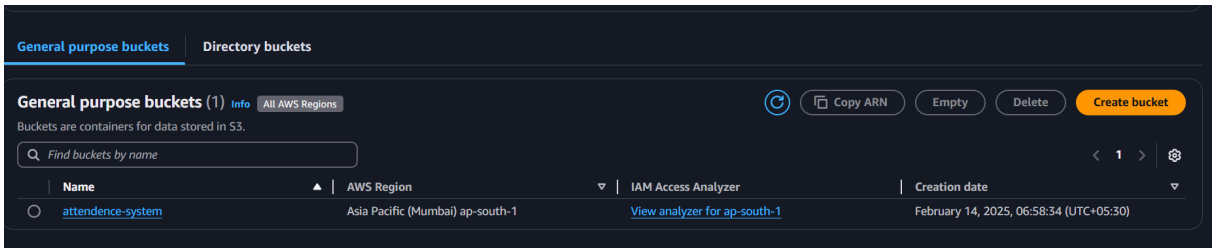
● To add data in DynamoDB:

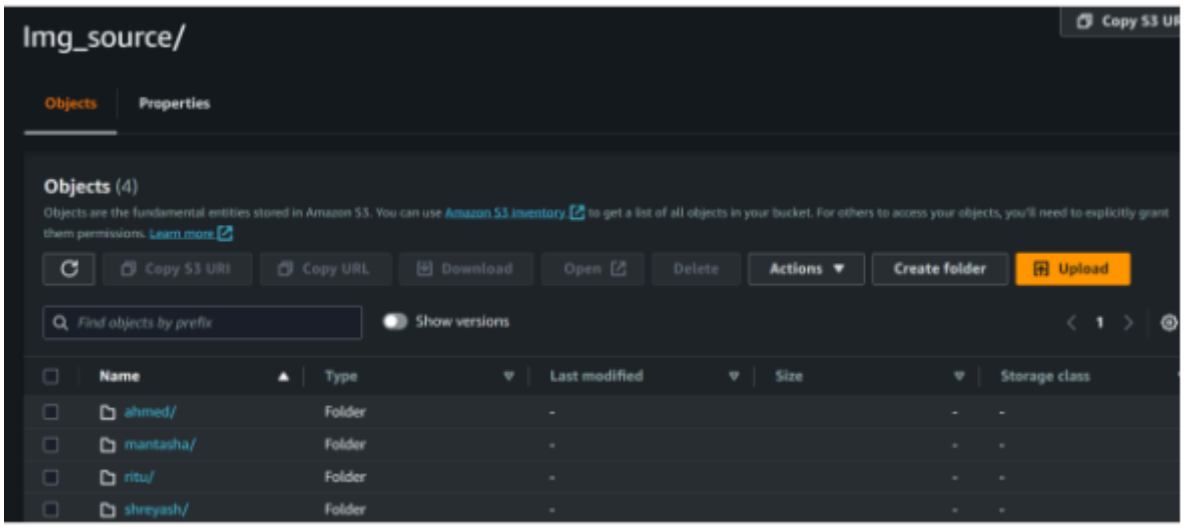
Initially, we'll generate a training dataset on our local system, capturing a minimum of 50-60 facial images for each individual.



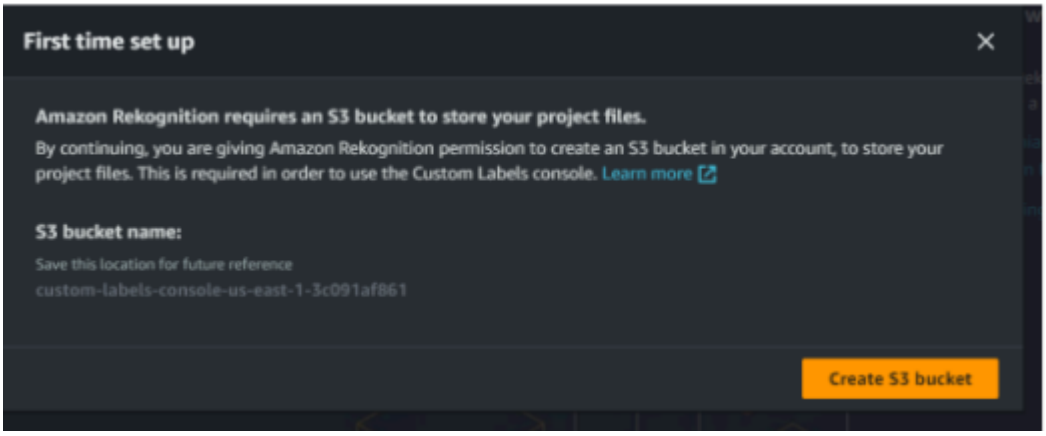
Next, we'll establish an S3 Bucket in AWS within the preferred region. Subsequently, we'll transfer the folder containing images from our system to the S3 Bucket.





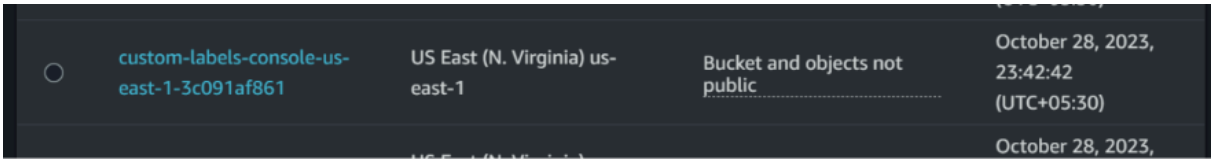


- After successful uploading of data in S3, navigate to the AWS Rekognition Service within the region where you've established the S3 bucket.
  - Access the "Use Custom Labels" section, initiate by clicking "Get Started."

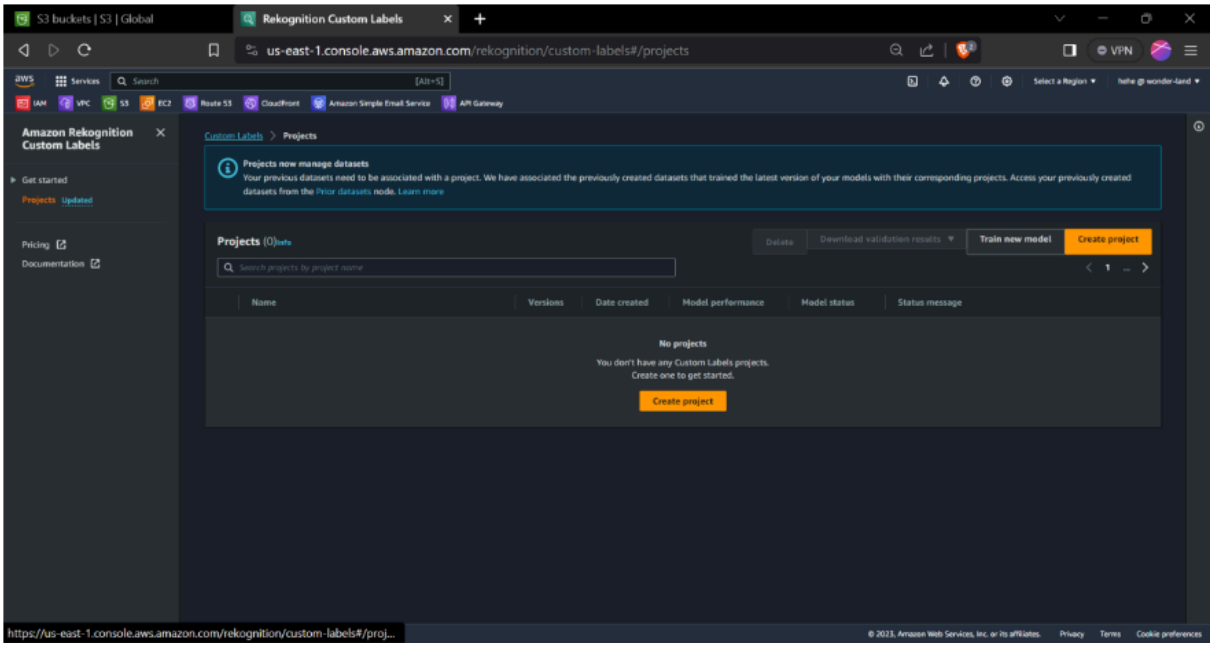


4

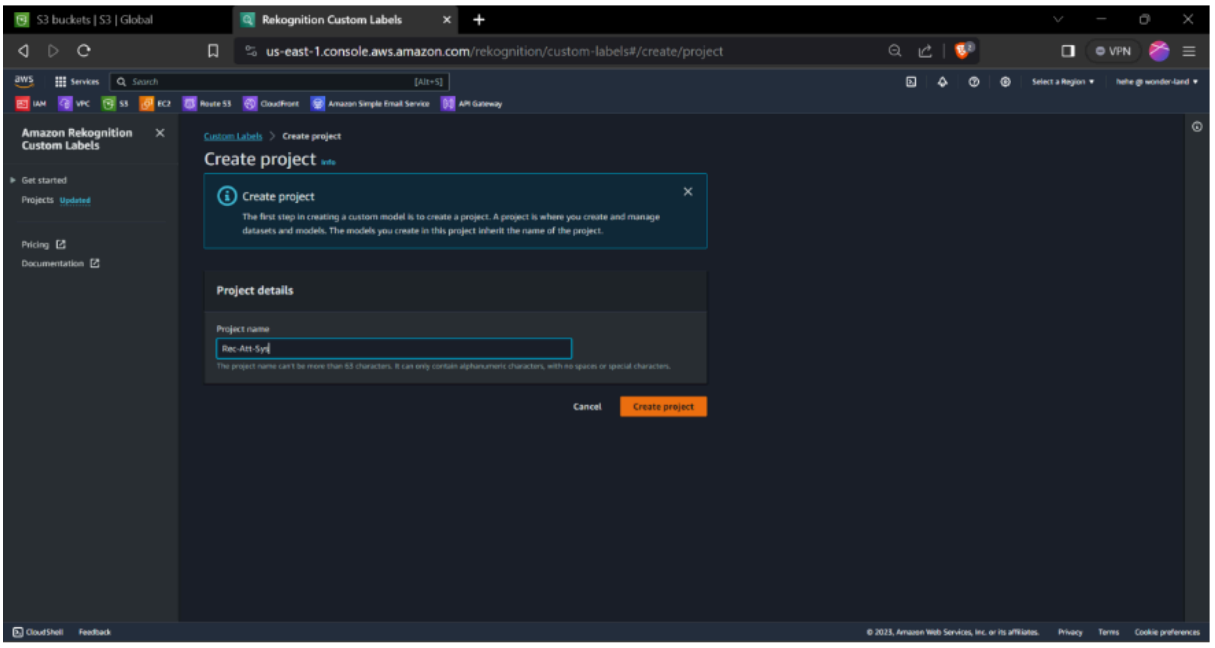
- For the initial setup, you'll be prompted to generate an S3 bucket with a specific name, like '**custom-labels-console-us-east-1-73d5fda5c8**.'
- Recognition functionality is contingent upon existence of this dedicated S3 bucket. Process by selecting “Create S3 bucket.”



Under Custom Labels, navigate to the Project section and initiate the creation process by selecting "Create Project."

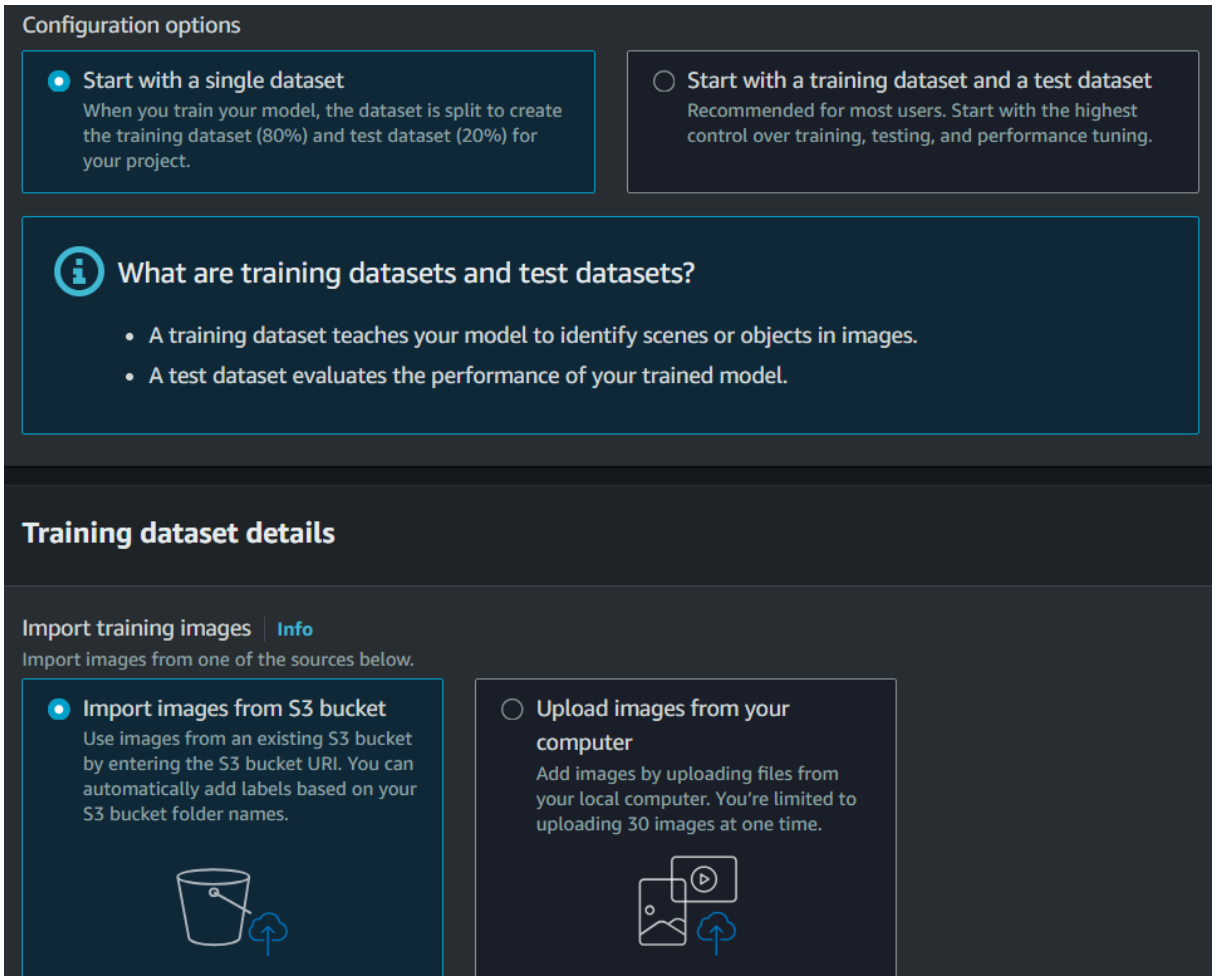


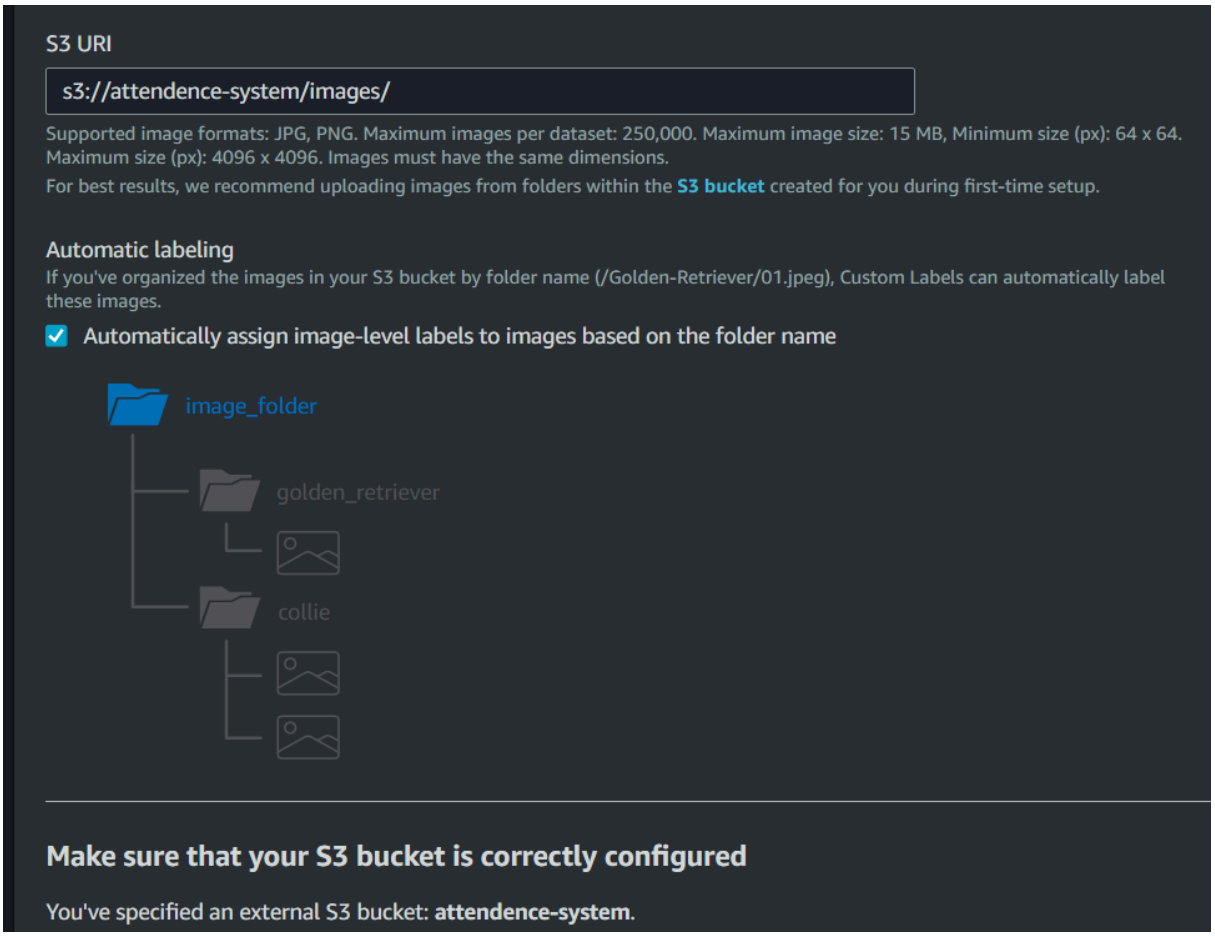
Provide a Project Name of your choice.



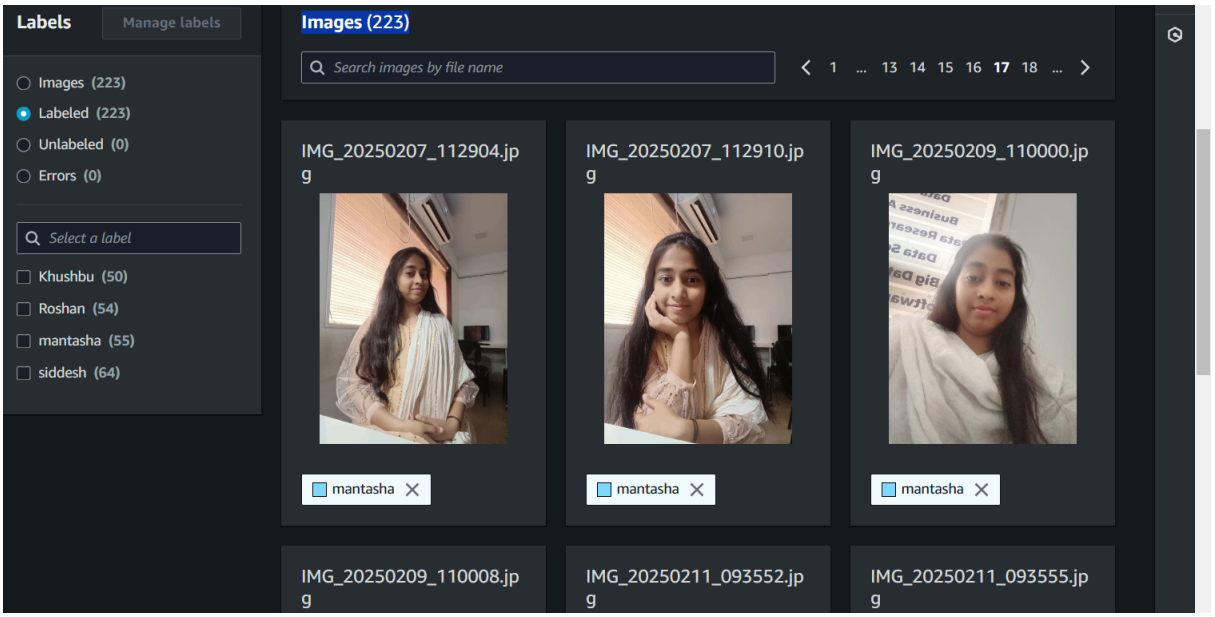
Select "Create Dataset" → Opt for "Start with Single Dataset" → Choose "Import images from S3" → Input the URI of the S3 bucket (The bucket created which we have uploaded data in).

Ensure to check the box for Automatic labelling before clicking on "Create Dataset."

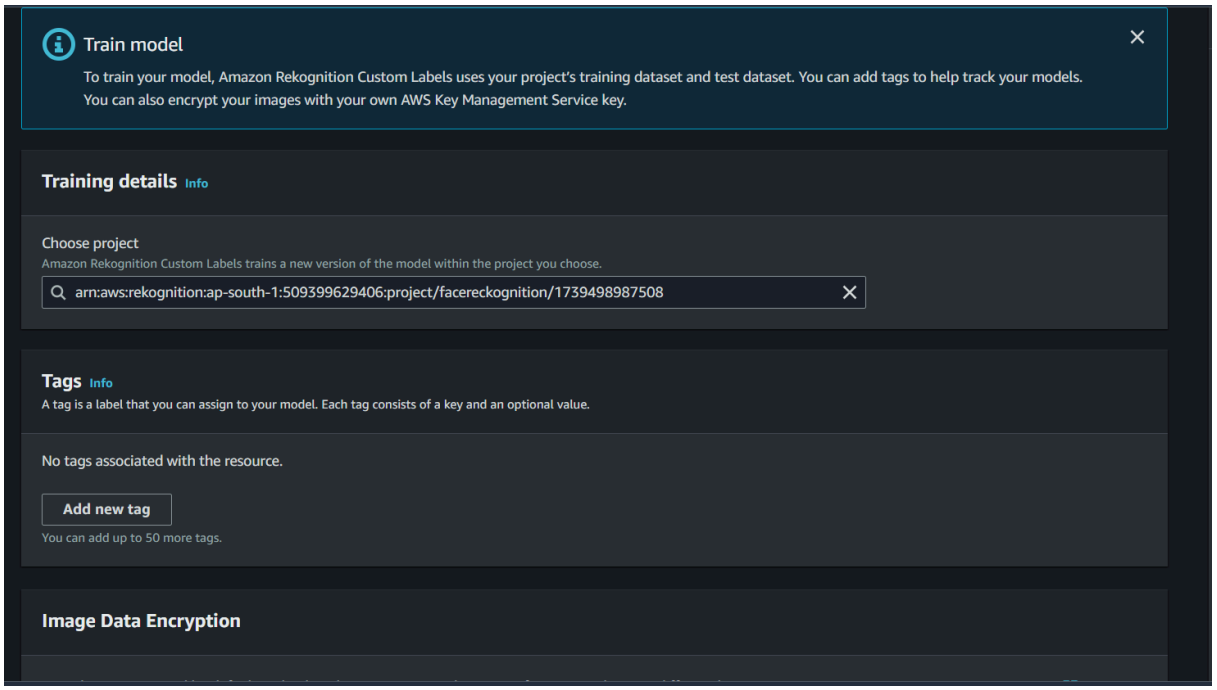


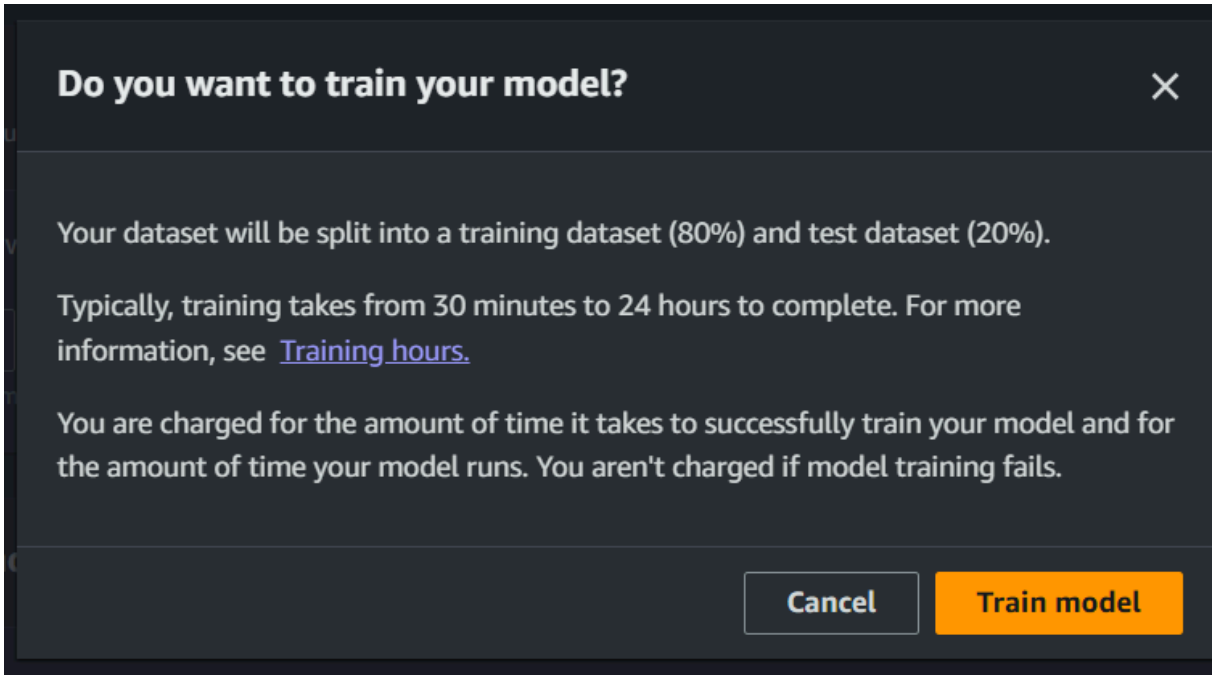


Initiate the labelling process by selecting "Start Labelling," pick the photos for labelling, opt for the "Draw Bounding Boxes" method, and complete the process by clicking "Done."

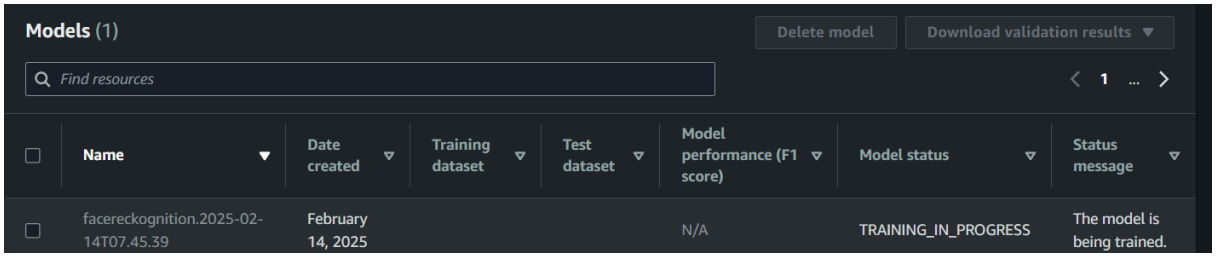


Now, proceed to train the model by selecting the project from the dropdown menu. Add tags if necessary, and finally, click on "Train Model" to initiate the training process.





Progress of the training can be seen under “Models” section. The process duration ranges from 30 minutes to 24 hours, depending on the number of images



***Note ***

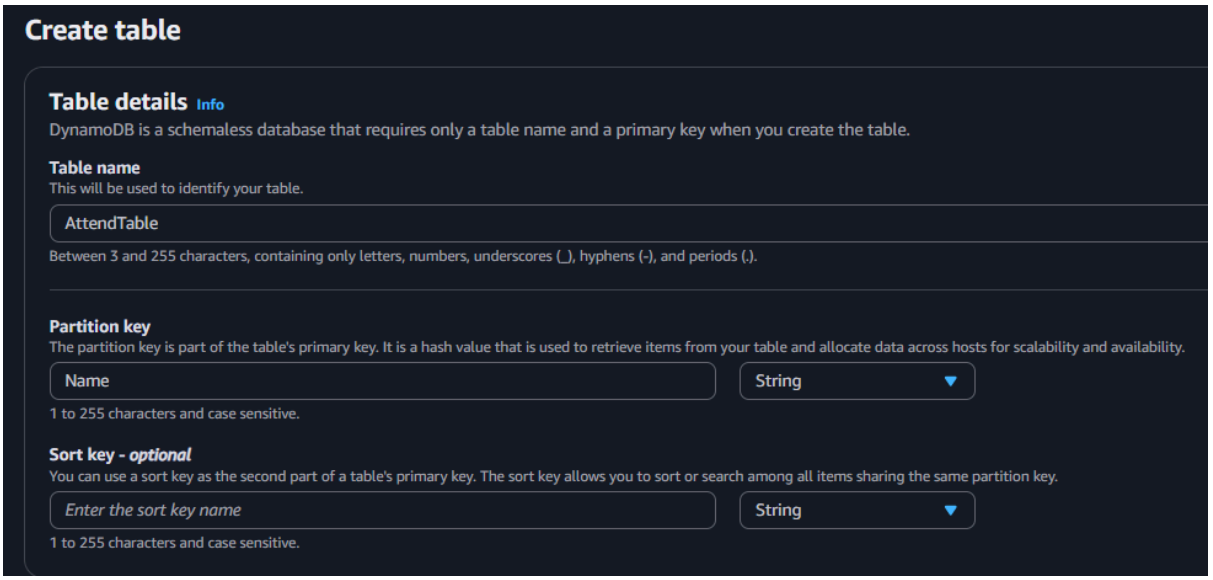
Using only one or two persons' images may result in an error "Too few labels in manifest files," so ensure a minimum of 5 persons' images.

Additionally, for accurate detection, it is recommended to use at least 50 images for each person if you are considering to take between 5-15 images.

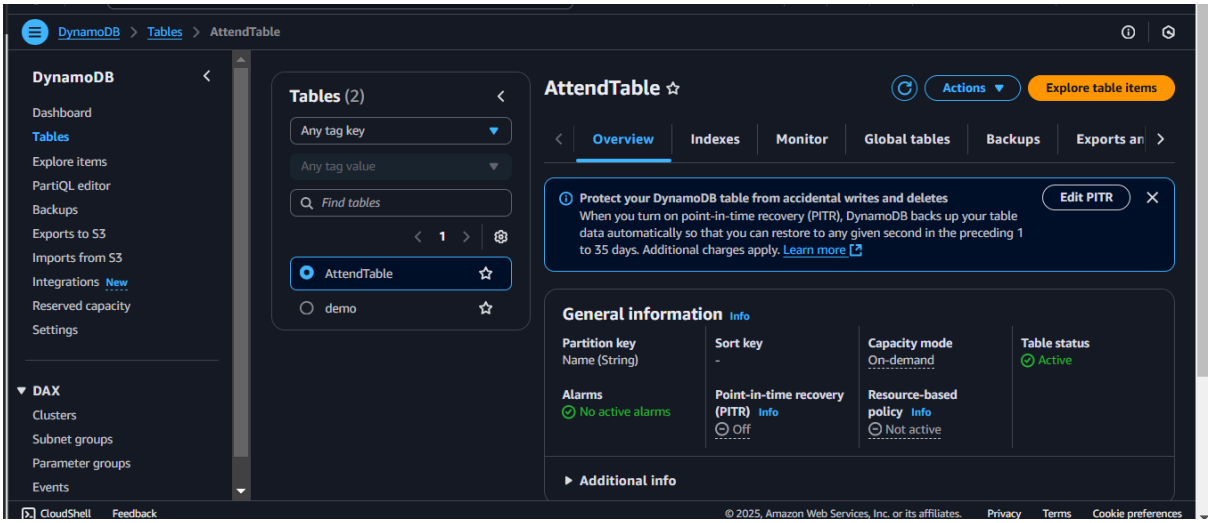
Patience is required while the model training process is to complete.

BACKEND

Initially, we will establish a table in DynamoDB.

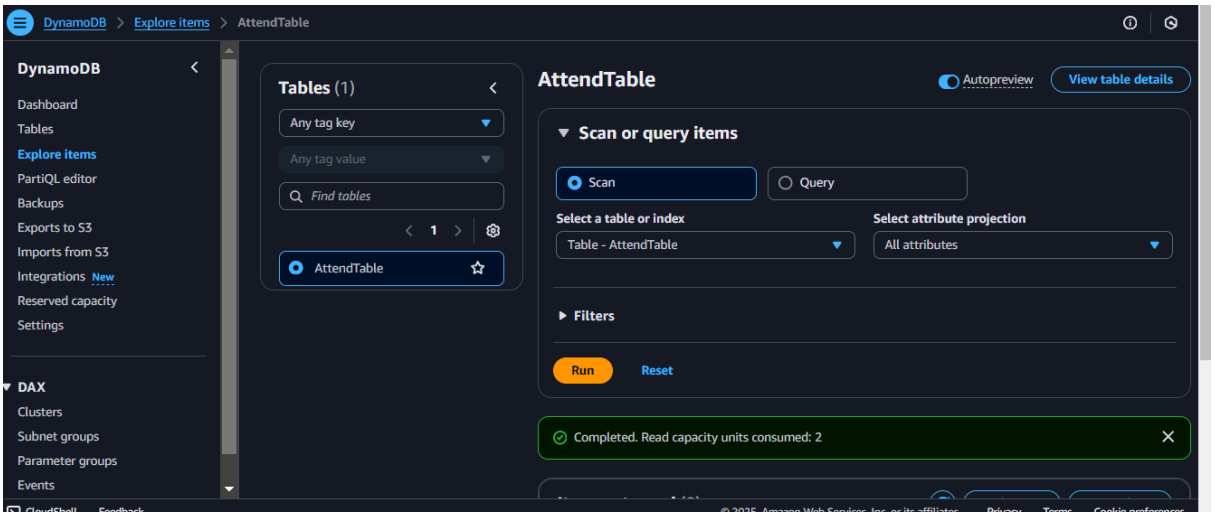


Provide a name and designate the partition key as "Name," leaving the remaining settings as default. Click on "Create" to proceed.



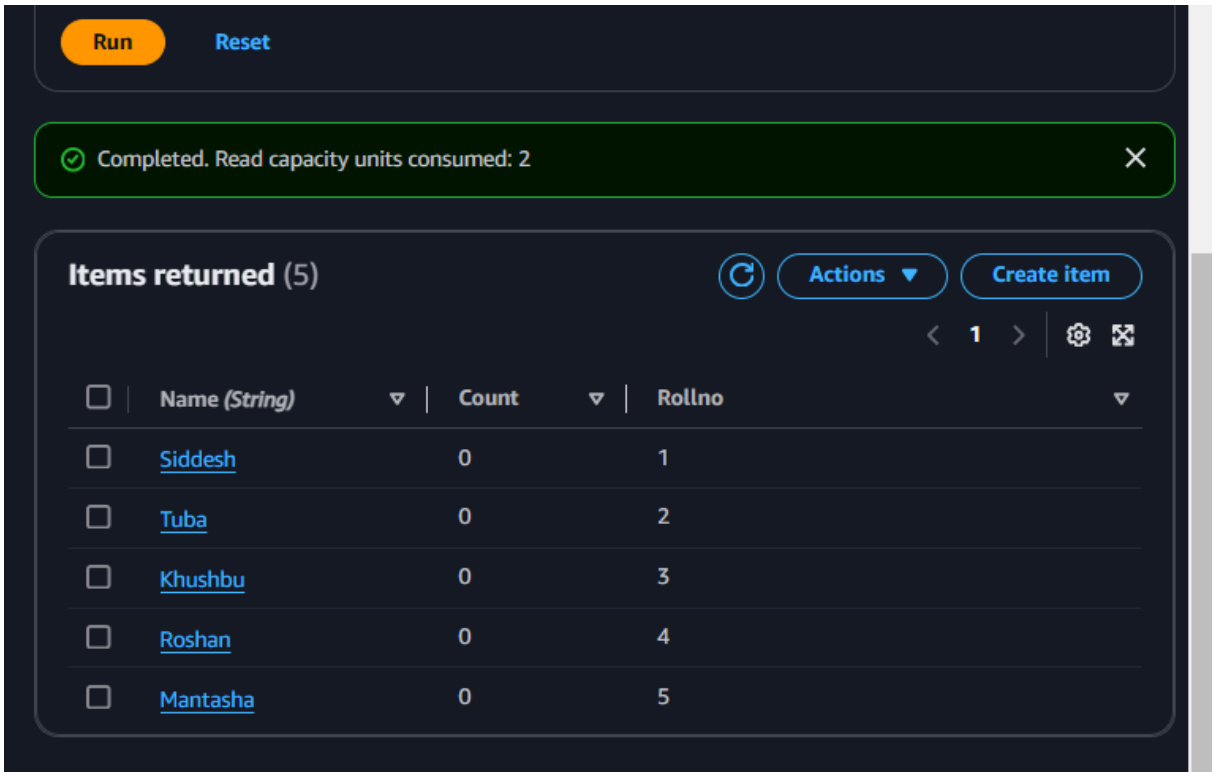
9

After the table is active, navigate to it and select "Explore Items."

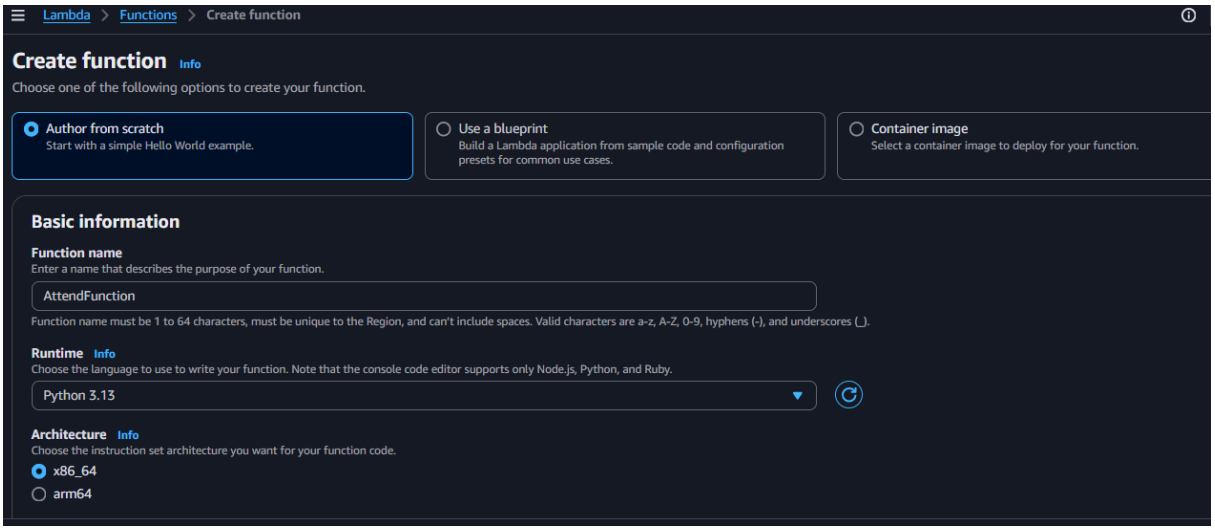


In this section, generate items corresponding to the labels (student names). For attributes,

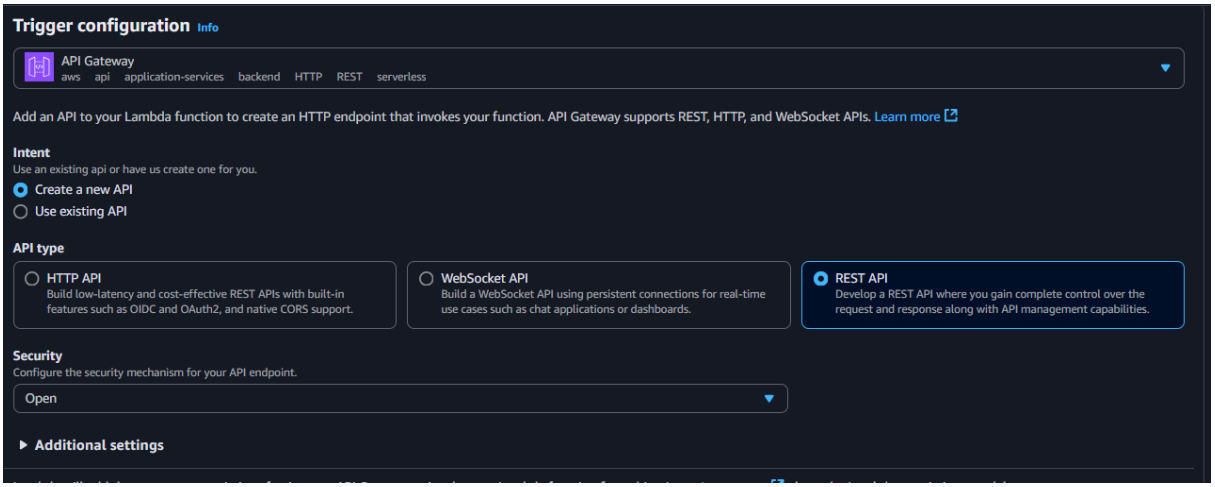
assign "Name" as string, "Rollno" as number, and "Count" as number, with the value set to 0 for each item.



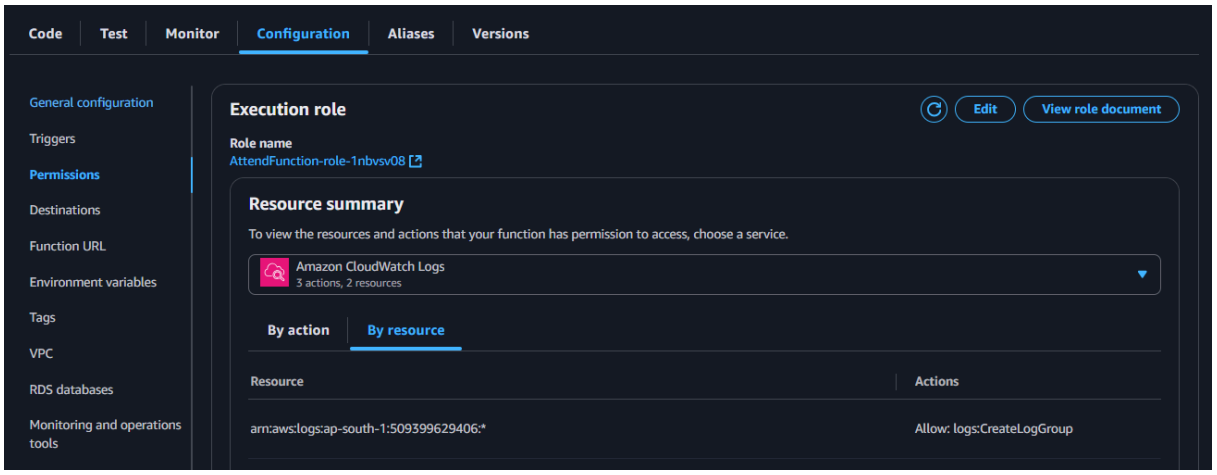
Once the dynamodb table is ready we will move forward to the lambda functions here we will create a function to update the dynamodb whenever an attendance is marked

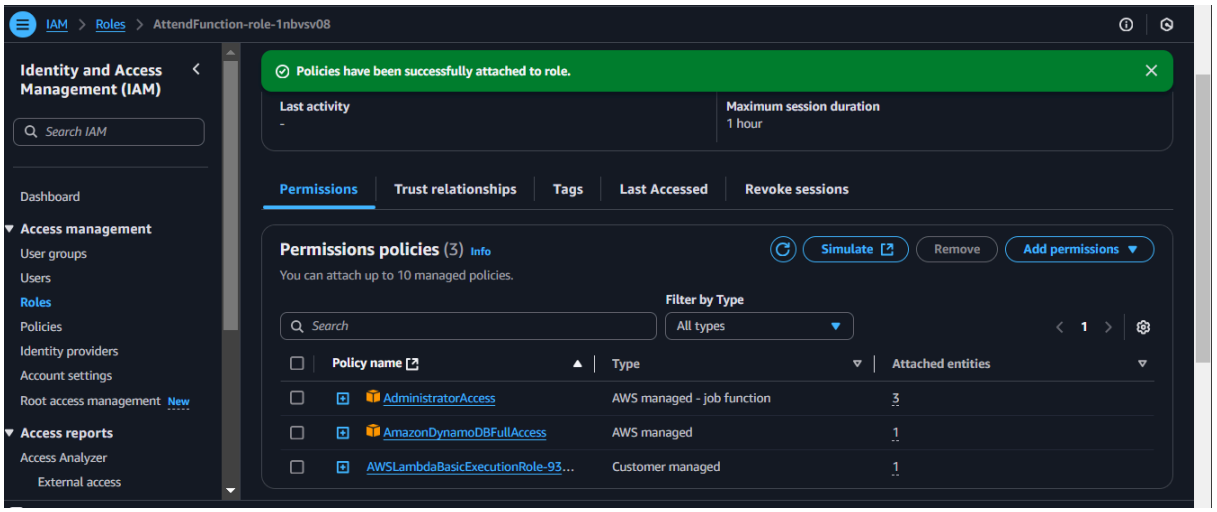


After creating the function we will add a trigger select api gateway , create new api, rest api, open and create



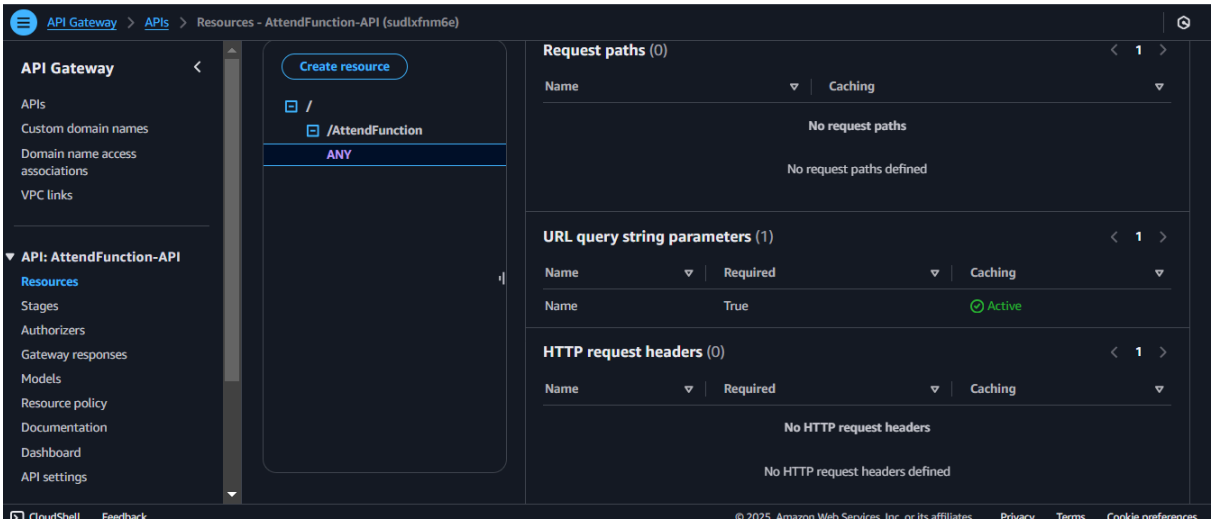
Then go to roles in the configuration permission goto iam role and edit these role





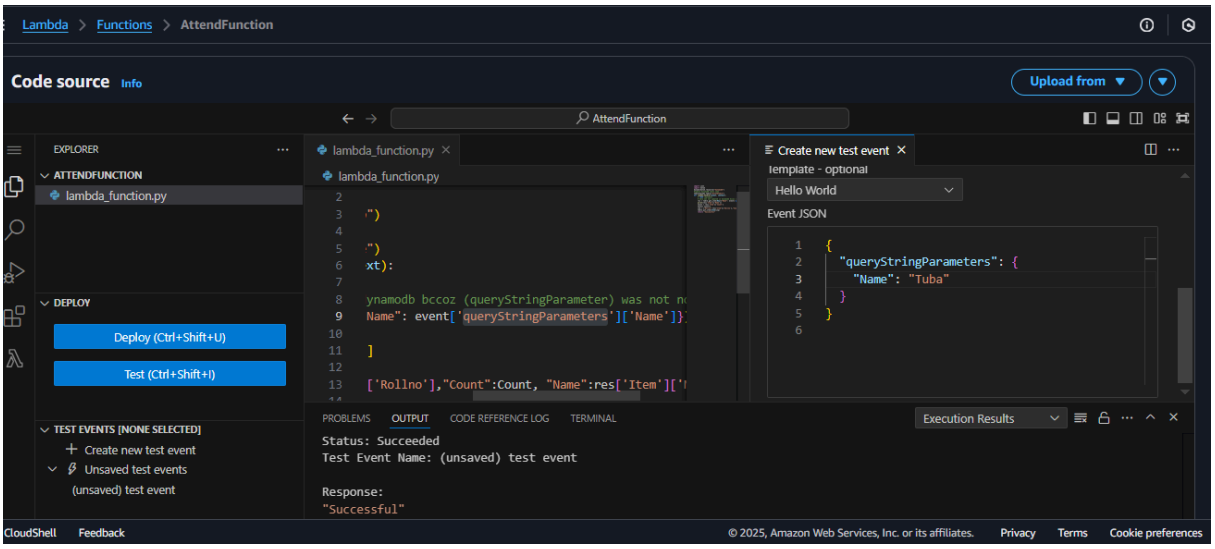
Here add <u>dynamodbfull</u> access in the permissions

Now give the lambda function python code in the lambda code
Then go to api gateway url from configuration -> triggers
Here add the **<u>querystringparamters</u>** in the method and deploy the api



Here add the query string parameter as name

Now you can test the code by creating an event

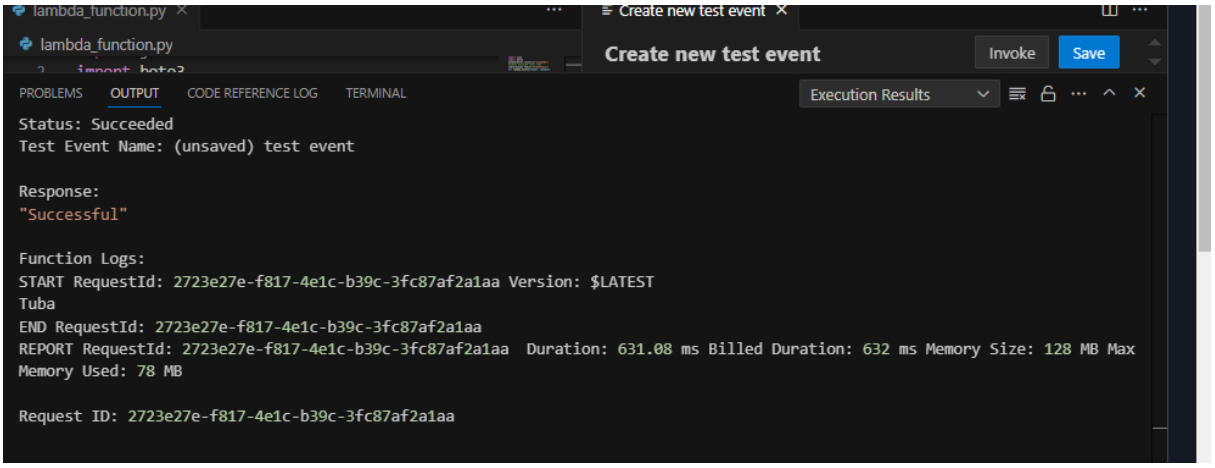


You must see the succeeded status and this will update your table attributes counts as you have given the test code(json)

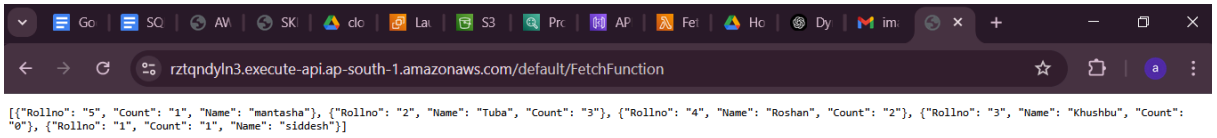
```
{
  "queryStringParameters": {
    "Name": "Tuba"
  }
}  # add these while testing your lambda function
```


Now same like that go to functions and create a function for fetching the data into the website to show the attendance for this use the http api when creating a trigger and give the code and test the code

It must show success and then go to api gateway endpoint url to see the results of fetched data.



Here you should see something like this



Attend function code:

```
import json
import boto3
dynamo=boto3.resource("dynamodb")
#update DynamoDB table name
table=dynamo.Table("AwsAttendTable")
def lambda_handler(event, context):
    # TODO implement
    # Data was not updated in dynamodb bccoz (queryStringParameter) was not not
    there.. res = table.get_item(Key={"Name": event['queryStringParameters']['Name']})
    print(res['Item']['Name'])
    Count = res['Item']['Count']
    Count= Count+1
    inp = {"Rollno":res["Item"]['Rollno'],"Count":Count, "Name":res['Item']['Name']}
    table.put_item(Item=inp)
    return "Successful"
```

FetchFunction:

```
import json
import boto3
dynamo=boto3.resource("dynamodb")
#update DynamoDB Table Name
table=dynamo.Table("AwsAttendTable")
def lambda_handler(event, context):
    # TODO implement
    response=table.scan()
    print(response)
    data= []
    for item in response['Items']:
      item['Rollno'] = str(item['Rollno'])
```



```
      item['Count'] = str(item['Count'])
      data.append(item)
    print(data)
    return{ "statusCode": 200,
    "headers": {
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Credentials':True,
      'Access-Control-Allow-Methods': 'GET, POST, PUT,
    DELETE', 'Access-Control-Allow-Headers': 'Content-Type,
    Authorization', },
    "body": json.dumps(response["Items"])}
```

Main.py

This code facilitates the capture of an image and its storage in an S3 bucket.

```
# Import the OpenCV library
import cv2
```

```
# Define a video capture object, which connects to your camera (camera index 0)
vid = cv2.VideoCapture(0)

# Flag to indicate whether to capture an image
capture_image = False

# Start an infinite loop to continuously capture frames
while True:
    # Capture a video frame from the camera
    ret, frame = vid.read()

    # Display the captured frame in a window labeled 'frame'
    cv2.imshow('frame', frame)

    # Check if the spacebar (' ') key is pressed
    if cv2.waitKey(1) & 0xFF == ord(' '):
        # If the spacebar is pressed, save the frame as an image named
        'captured_image.jpg' cv2.imwrite('captured_image.jpg', frame)
        print("Image captured!")
        # Reset the flag to indicate that the image has been captured
        capture_image = False

    # Check if the 'q' key is pressed to quit the program
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object to free up camera resources
vid.release()
```



```
# Close all OpenCV windows
cv2.destroyAllWindows()
```

attendance.py

This script continuously captures images on hourly intervals, sends these images to AWS Rekognition for face recognition, and updates student attendance using an associated API endpoint by marking attendance for recognized faces in the captured images. This process repeats for a total of 6 iterations, each separated by an hour.

```
import boto3
import requests
import datetime
import time

import cv2

# Credentials----------------------------------------------------------------------------
client = boto3.client('rekognition',
                      aws_access_key_id="AKIAXQKH4EDNZYI5VFGW",
                      aws_secret_access_key="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",
                      region_name='us-east-1')
```

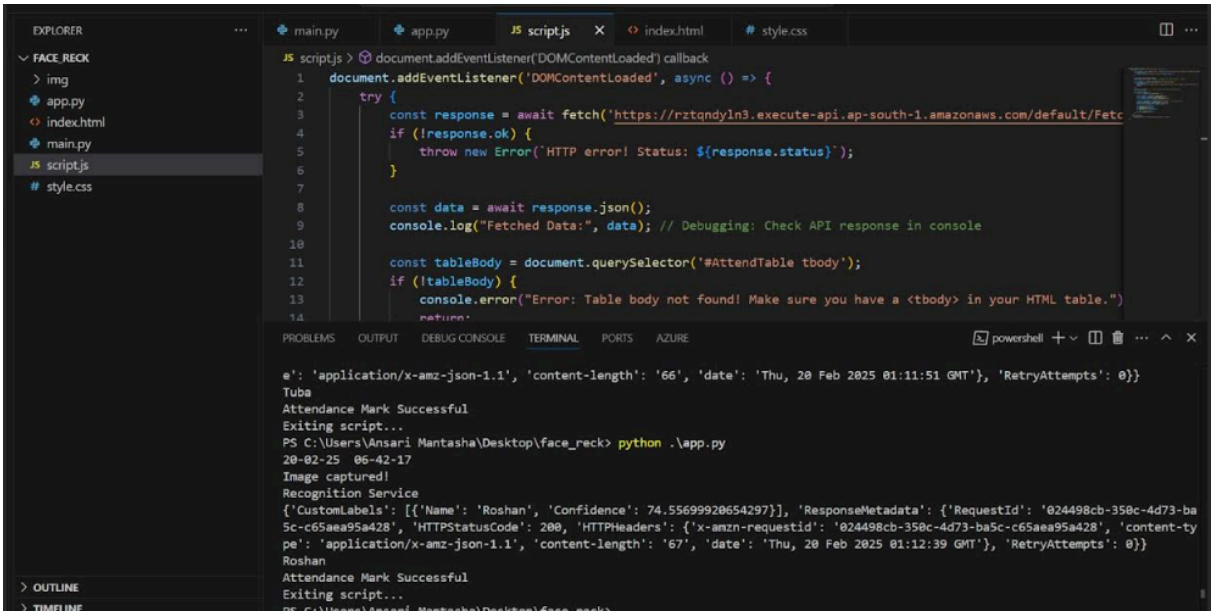
```
# Capture images for every 1 hour and store the image with current date and time
--------------------------------------------------------------------------------
for j in range(0, 6):
    current_time = datetime.datetime.now().strftime("%d-%m-%y %H-%M-%S ")
    print(current_time)
    camera = cv2.VideoCapture(0)

    while True:
        # Capture the video frame by frame
        ret, frame = camera.read()

        # Display the resulting frame
        cv2.imshow('frame', frame)
        # Check if the image needs to be captured
        if cv2.waitKey(1) & 0xFF == ord(' '):
            # Save the captured frame as an image
            cv2.imwrite('img/' + current_time + '.jpg', frame)
            print("Image captured!")
            # Reset the flag
            break

        # Check if the 'q' button is pressed to quit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            exit()

    del (camera)

    # Send the captured image to AWS S3
Bucket----------------------------------------------------------------------------
    clients3 = boto3.client('s3', aws_access_key_id="AKIAXQKH4EDNZYI5VFGW",
                            aws_secret_access_key="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",
region_name='us-east-1')
```

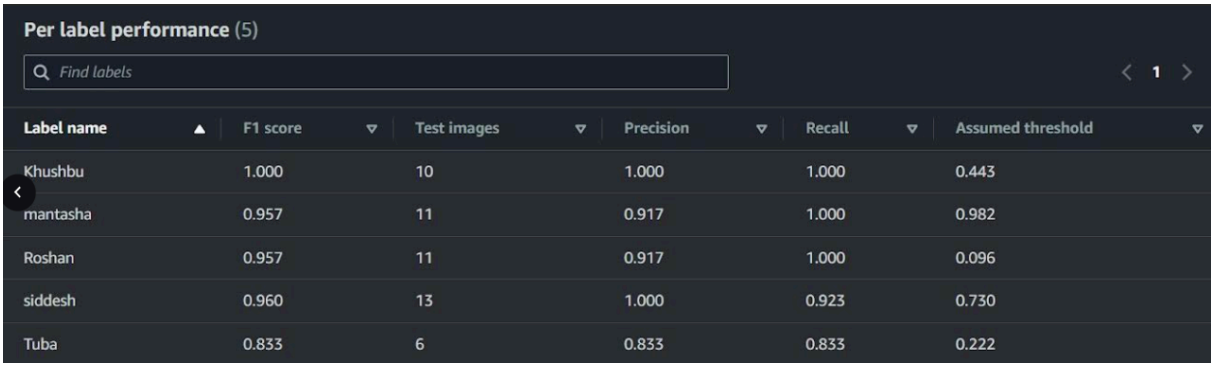


```
    # clients3.upload_file("Hourly Class Images/"+current_time+'.jpg', 'add your S3 bucket
name', current_time+'.jpg')

    clients3.upload_file("img/" + current_time + '.jpg', 'dun-dun', current_time + '.jpg')

    # Recognize students in captured image
--------------------------------------------------------------------------------
    image_path = 'img/' + current_time + '.jpg'
    with open(image_path, 'rb') as source_image:
        source_bytes = source_image.read()
    print(type(source_bytes))

    print("Recognition Service")
    response = client.detect_custom_labels(

        # Update the Recognition ARN with yours

ProjectVersionArn='arn:aws:rekognition:us-east-1:516083687643:project/Face-Rek-Att-Sys/version/Face-Rek
Att-Sys.2023-10-29T21.42.26/1698595945940',

        Image={
            'Bytes': source_bytes
        },
```

```
)

print(response)
if not len(response['Custom Labels']):
    print('Not identified')

else:
    str = response['Custom Labels'][0]['Name']
    print(str)

    # Update the attendance of recognized student in DynamoDB by calling the API

    url = "https://kx62h8ef40.execute-api.ap-south-1.amazonaws.com/Name/AttendTable?Name=" + str

    resp = requests.get(url)
    print("Attendance Mark Successful")
    if resp.status_code == 200:
        print("Success")

time.sleep(3600)
```

output :





Per label performance (5)

Find labels

1

Label name	F1 score	Test images	Precision	Recall	Assumed threshold
Khushbu	1.000	10	1.000	1.000	0.443
mantasha	0.957	11	0.917	1.000	0.982
Roshan	0.957	11	0.917	1.000	0.096
siddesh	0.960	13	1.000	0.923	0.730
Tuba	0.833	6	0.833	0.833	0.222

FRONTEND

For the frontend, we'll utilize HTML, CSS, and JavaScript. We'll create a table and use JavaScript to fetch data from the API, displaying it on our table.

index.html

Here we have written a simple code for table.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <table id="attendanceTable">
        <thead>
        <tr>
            <th>Name</th>
            <th>Roll No</th>
            <th>Count</th>
        </tr>
    </thead>
        <tbody></tbody>
    </table>
</body>
<script src="script.js"></script>
</html>
```


style.css

To design our table we have used CSS to make it more creative.

```
body {
    background-image: url('hehe.avif');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
}


table {
    font-family: Arial, sans-serif;
    border-collapse: collapse;
    width: 100%;
    background-color: rgba(255, 255, 255, 0.5);
    border-radius: 2px;
    border: #333;
     /* Adjust the alpha value for transparency */
    margin-left: 250px;
    margin-right: 250px;
}

th, td {
    border: 1px solid #333;
    padding: 10px;
    text-align: center;
}

th {
    background-color: rgba(0, 116, 217, 0.5); /* Header background color with transparency */
    color: white; /* Header text color */
}

td {
    background-color: rgba(242, 242, 242, 0.5); /* Cell background color with transparency */

    color: #333; /* Cell text color */
}
```

script.js

Here we have written code to fetch the data from API and display it in our webpage


```
document.addEventListener('DOMContentLoaded', () => {
fetch('https://0viq2ph712.execute-api.ap-south-1.amazonaws.com/default/FetchFunction').then(response
```

```
=> response.json())
.then(data => {
const tableBody = document.querySelector('#attendanceTable tbody');
data.forEach(student => {
const row = document.createElement('tr');
const rollNoCell = document.createElement('td');
const nameCell = document.createElement('td');
const attendanceCell = document.createElement('td');
rollNoCell.text Content = student.Name;
nameCell.textContent = student.Rollno;
attendanceCell.textContent = student.Count;
row.appendChild(rollNoCell);
row.appendChild(nameCell);
row.appendChild(attendanceCell);
tableBody.appendChild(row);
});
})
.catch(error => {
console.error(error);
});
});
```

Output:

Name	Roll No	Count
5	mantasha	1
2	Tuba	2
4	Roshan	2
3	Khushbu	0
1	siddesh	1

REFERENCE

− https://github.com/CriMenio/Face-Detection-Attendance-Management-System −

https://aws.amazon.com/rekognition/custom-labels-features/ −