

"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.

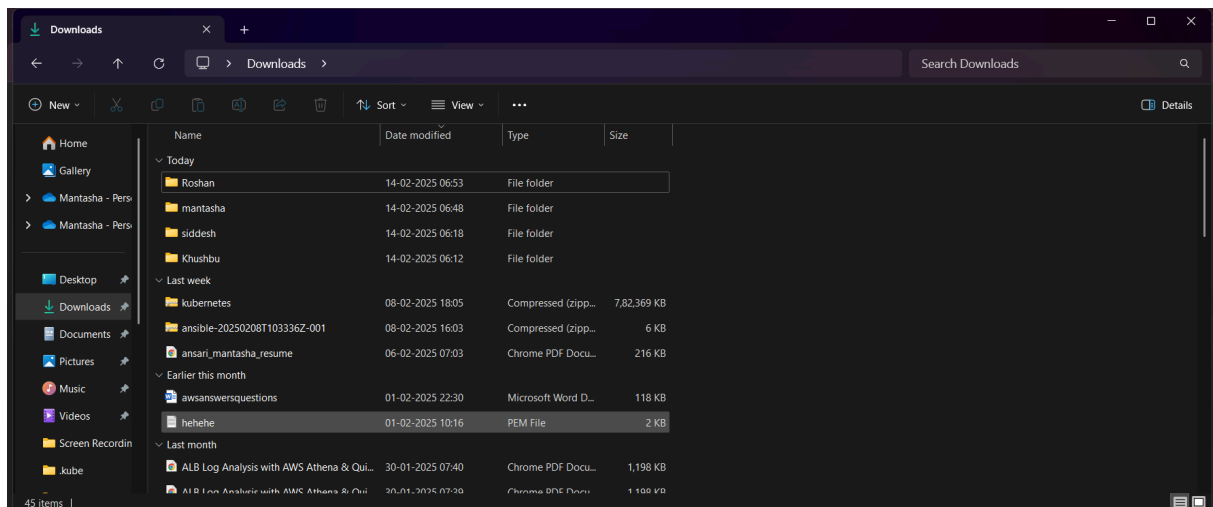
BLOCK DIAGRAM



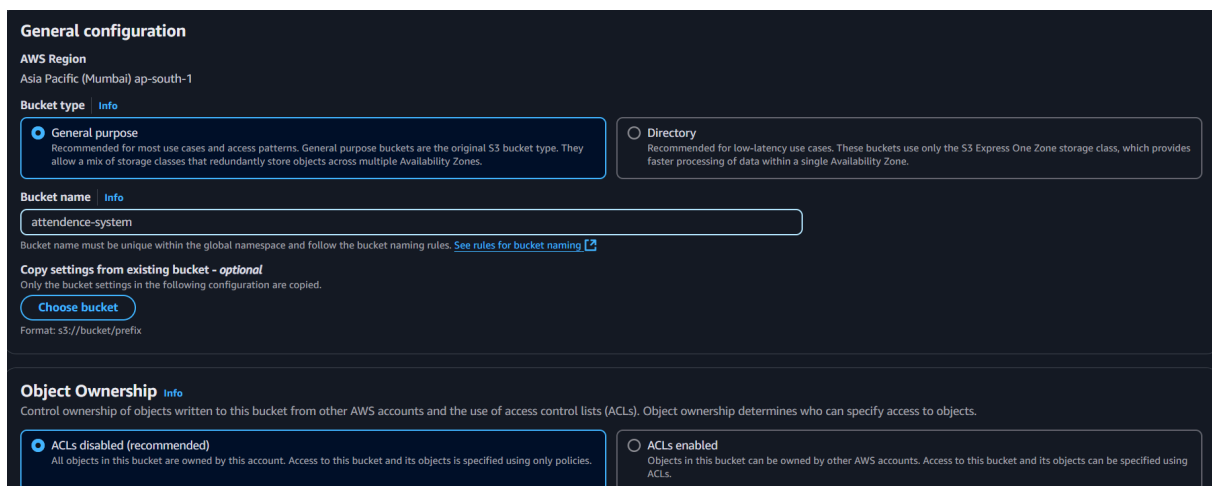
IMPLEMENTATION

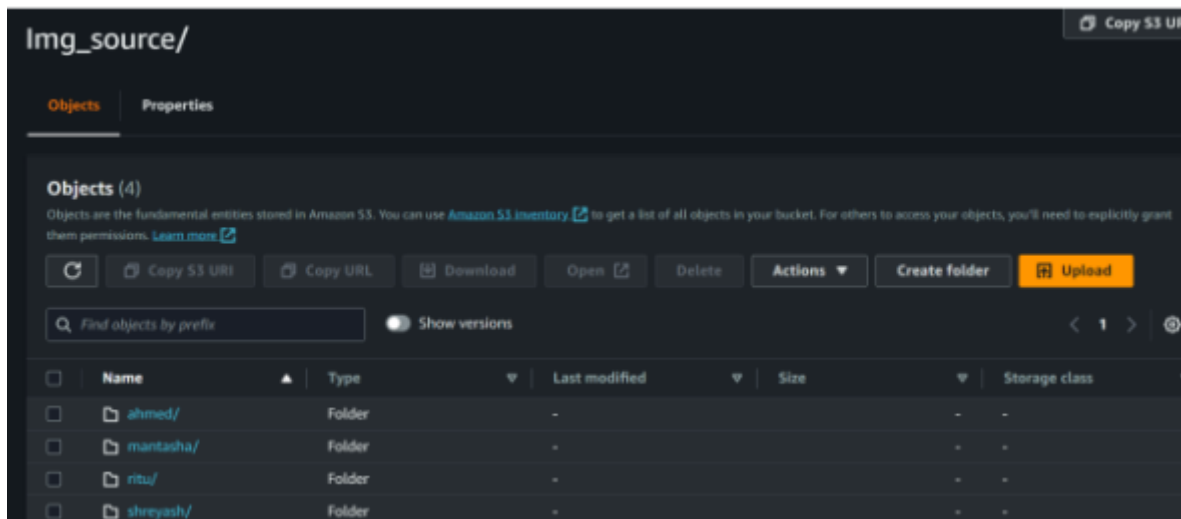
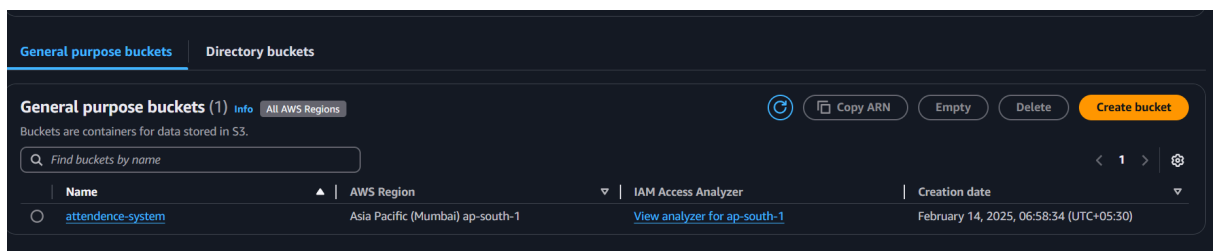
● To add data in DynamoDB:

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

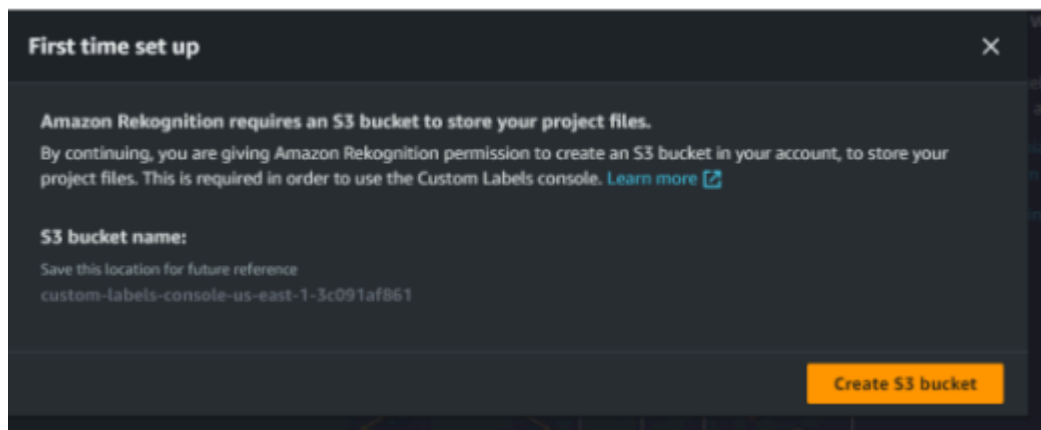


Next, we'll establish an S3 Bucket in AWS within the preferred region. Subsequently, we'll transfer the folder containing images from our system to the S3 Bucket.





- 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."

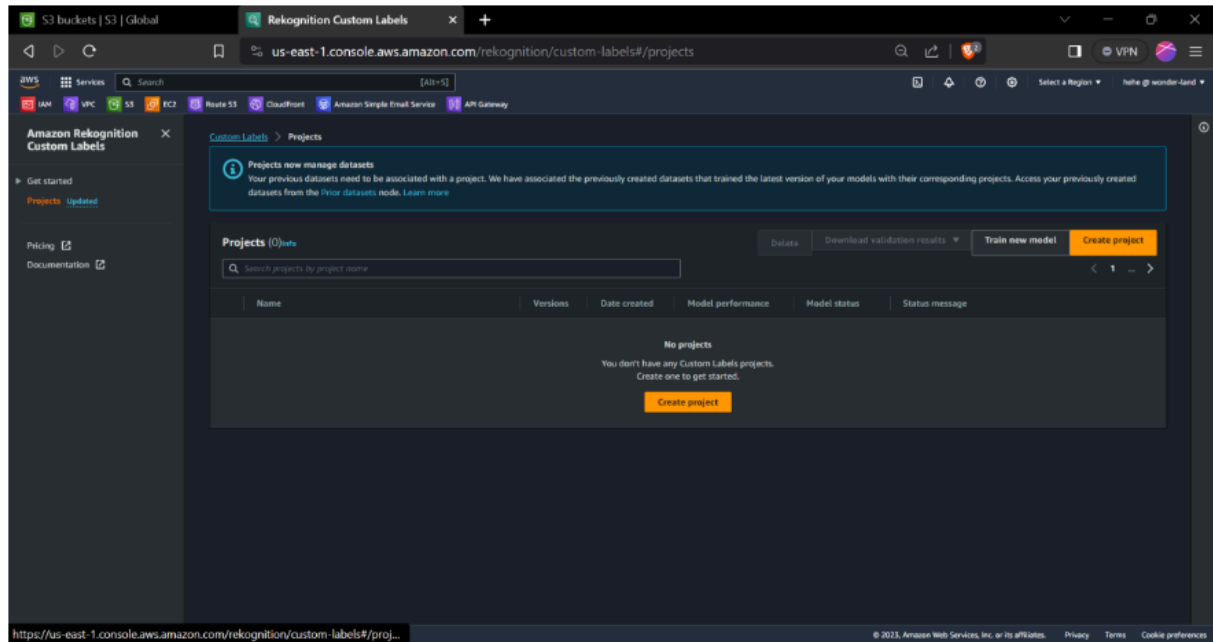


4

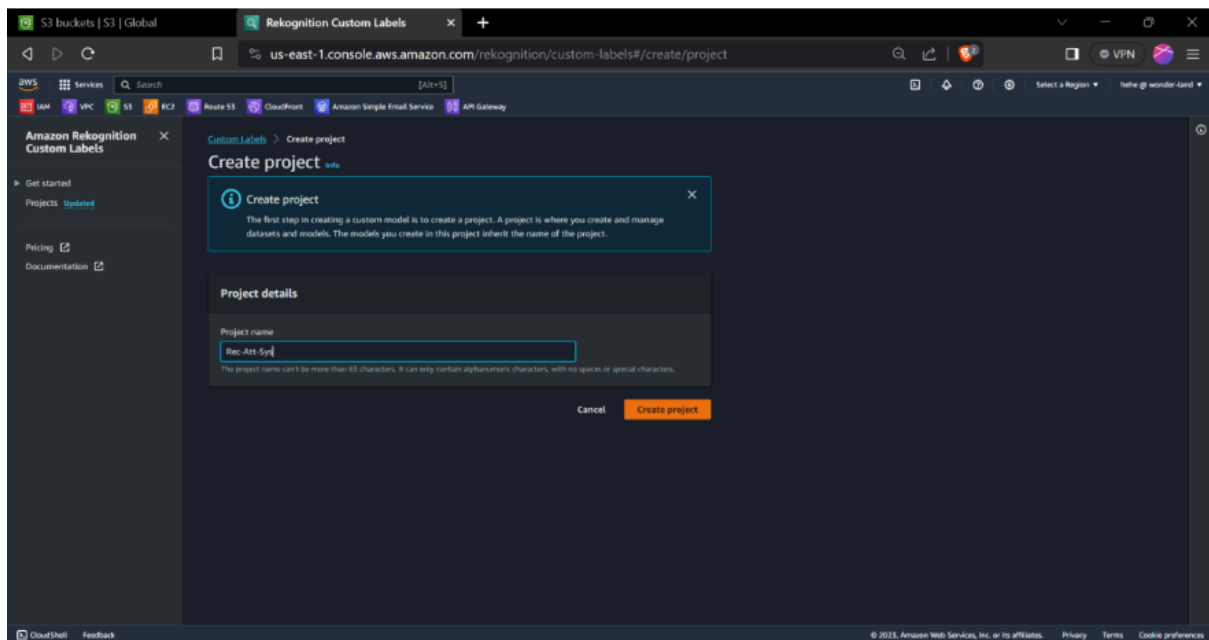
- For the initial setup, you'll be prompted to generate an S3 bucket with a specific name, like '**custom-labels-console-us-east-1-73d5fda5c8.**'
- Recognition functionality is contingent upon existence of this dedicated S3 bucket. Process by selecting “Create S3 bucket.”

<input type="radio"/>	custom-labels-console-us-east-1-3c091af861	US East (N. Virginia) us-east-1	Bucket and objects not public	October 28, 2023, 23:42:42 (UTC+05:30)
<input type="radio"/>	custom-labels-console-us-east-1-73d5fda5c8	US East (N. Virginia) us-east-1	Bucket and objects not public	October 28, 2023, 23:42:42 (UTC+05:30)

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

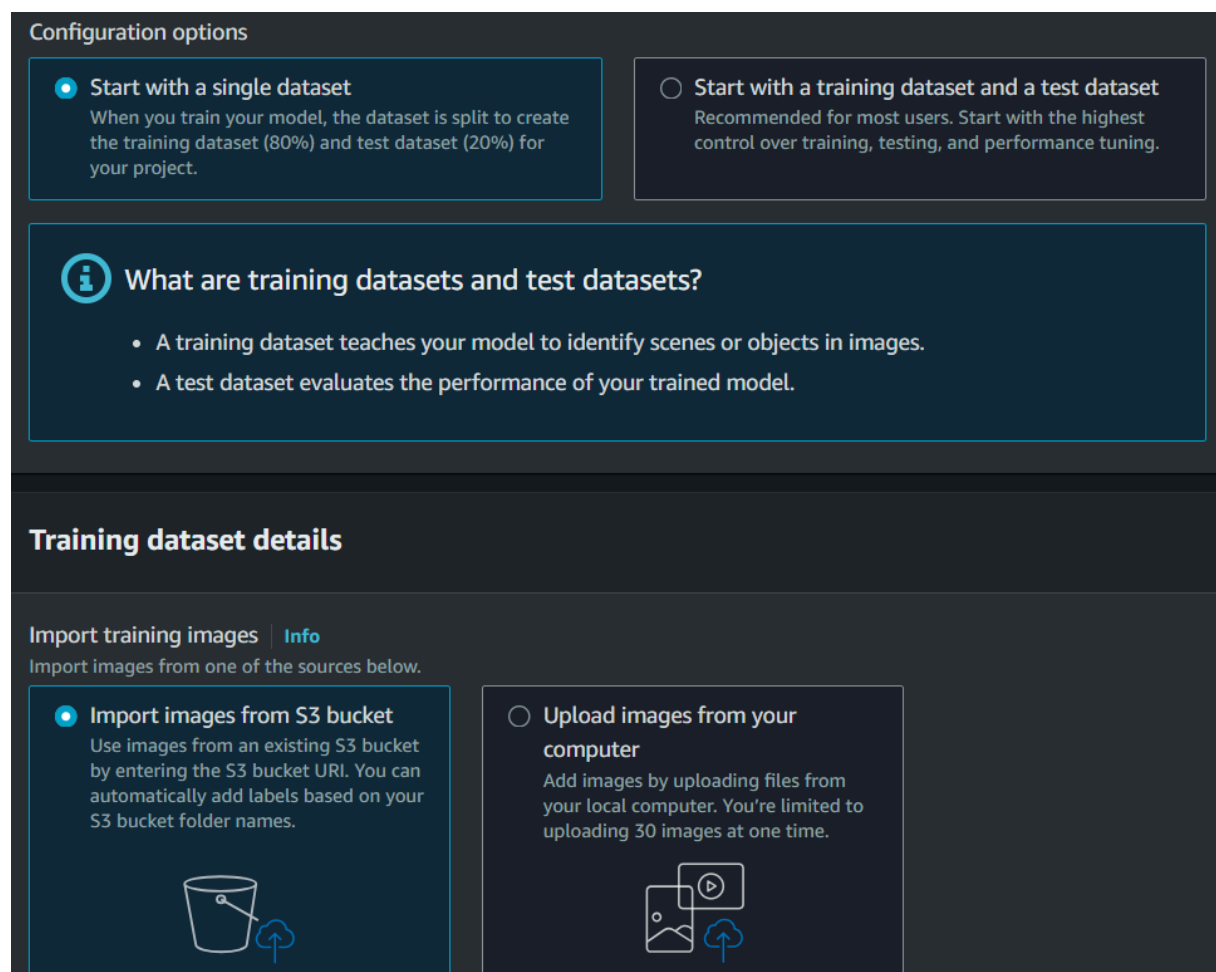


Provide a Project Name of your choice.



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."



S3 URI

`s3://attendance-system/images/`

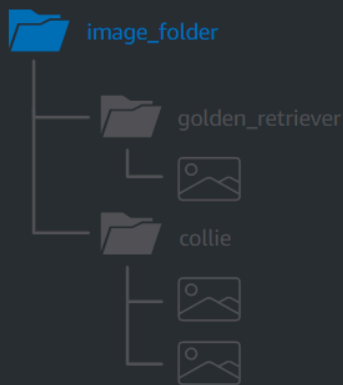
Supported image formats: JPG, PNG. Maximum images per dataset: 250,000. Maximum image size: 15 MB, Minimum size (px): 64 x 64. Maximum size (px): 4096 x 4096. Images must have the same dimensions.

For best results, we recommend uploading images from folders within the **S3 bucket** created for you during first-time setup.

Automatic labeling

If you've organized the images in your S3 bucket by folder name (/Golden-Retriever/01.jpeg), Custom Labels can automatically label these images.

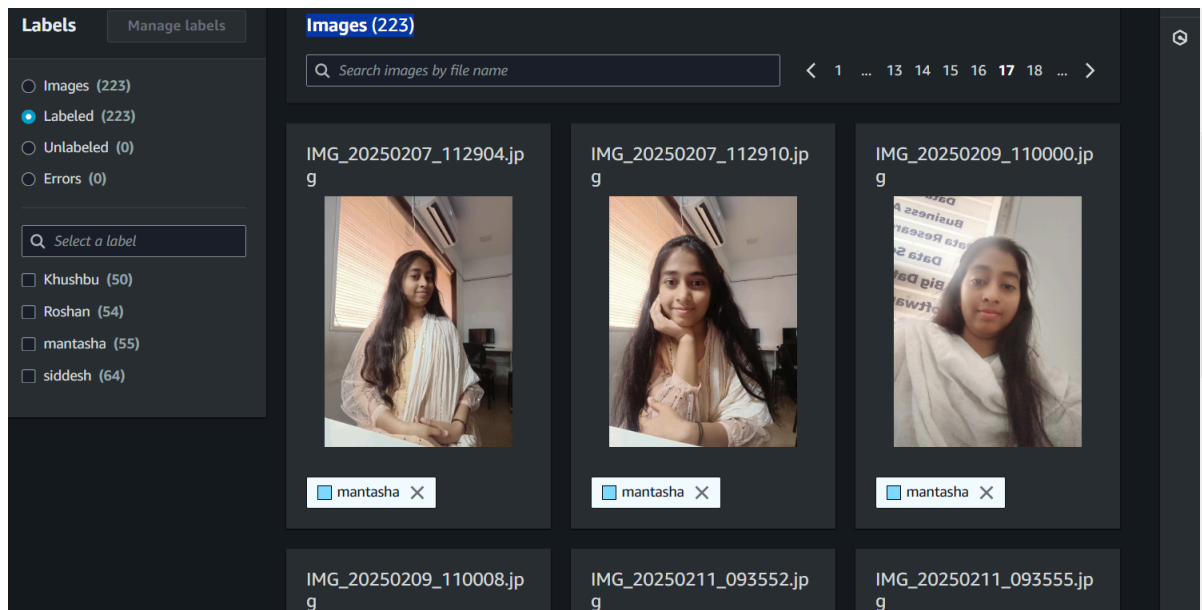
- ☒ Automatically assign image-level labels to images based on the folder name



Make sure that your S3 bucket is correctly configured

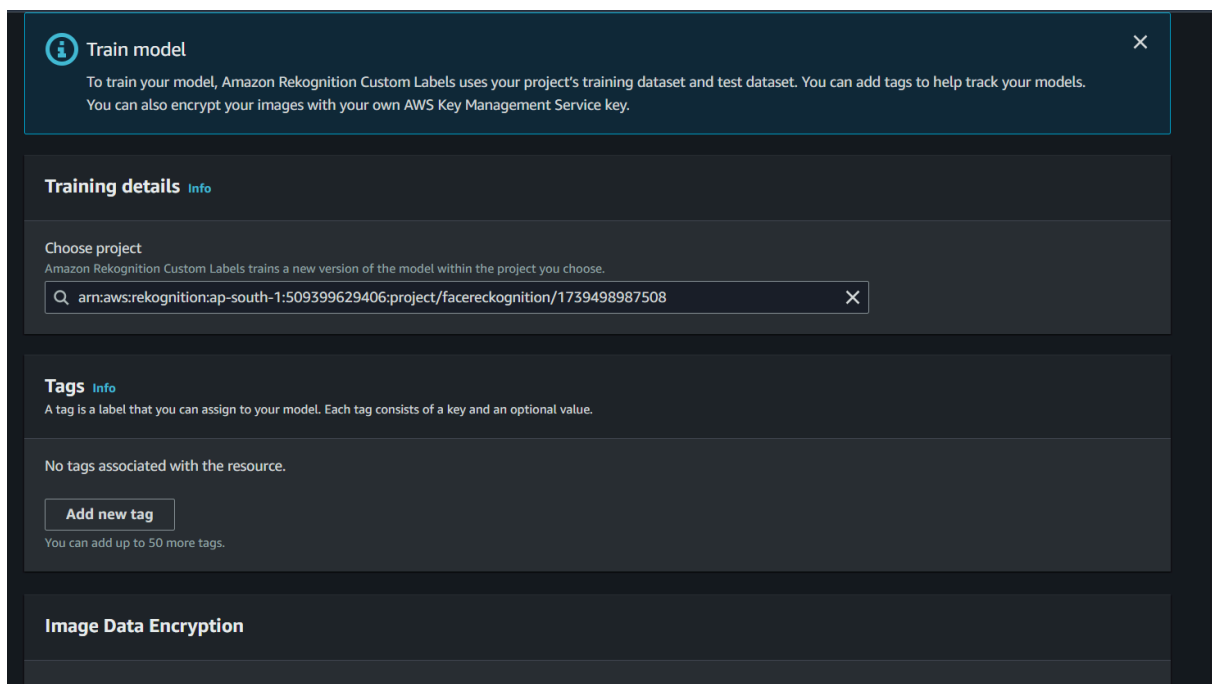
You've specified an external S3 bucket: **attendance-system**.

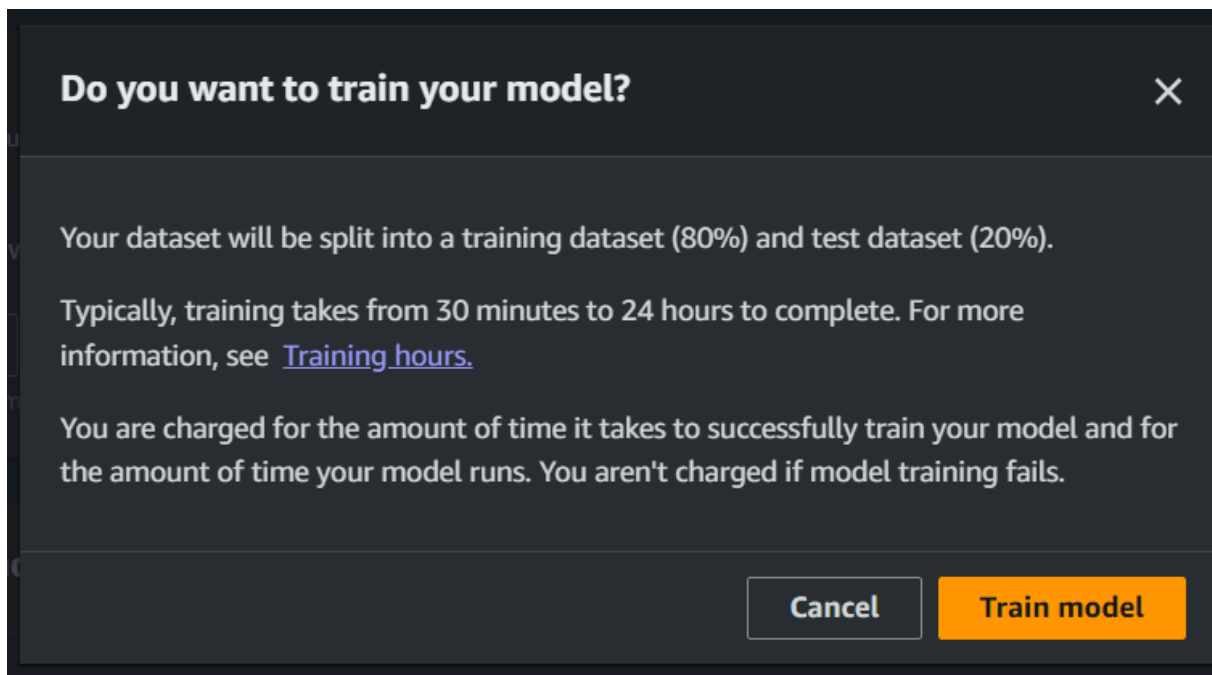
Initiate the labelling process by selecting "Start Labelling," pick the photos for labelling, opt for the "Draw Bounding Boxes" method, and complete the process by clicking "Done."



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.

7





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

Models (1)

Delete model

Download validation results ▾

Find resources

< 1 ... >

<input type="checkbox"/>	Name ▾	Date created ▾	Training dataset ▾	Test dataset ▾	Model performance (F1 score) ▾	Model status ▾	Status message ▾
<input type="checkbox"/>	facerecognition.2025-02-14T07.45.39	February 14, 2025			N/A	TRAINING_IN_PROGRESS	The model is being trained.

***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

8

Initially, we will establish a table in DynamoDB.

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

AttendTable

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

Name String

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name String

1 to 255 characters and case sensitive.

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

The screenshot shows the AWS Management Console for DynamoDB. The left sidebar contains navigation options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main content area shows the 'AttendTable' configuration page. The 'Tables (2)' list on the left includes 'AttendTable' and 'demo'. The 'AttendTable' configuration page has tabs for Overview, Indexes, Monitor, Global tables, Backups, and Exports and imports. The 'Overview' tab is selected, showing a warning about PITR and general information. The 'General information' section shows the partition key as 'Name (String)', the sort key as '-', the capacity mode as 'On-demand', and the table status as 'Active'. The 'Alarms' section shows 'No active alarms'. The 'Point-in-time recovery (PITR)' is set to 'Off'. The 'Resource-based policy' is set to 'Not active'. The 'Additional info' section is expanded.

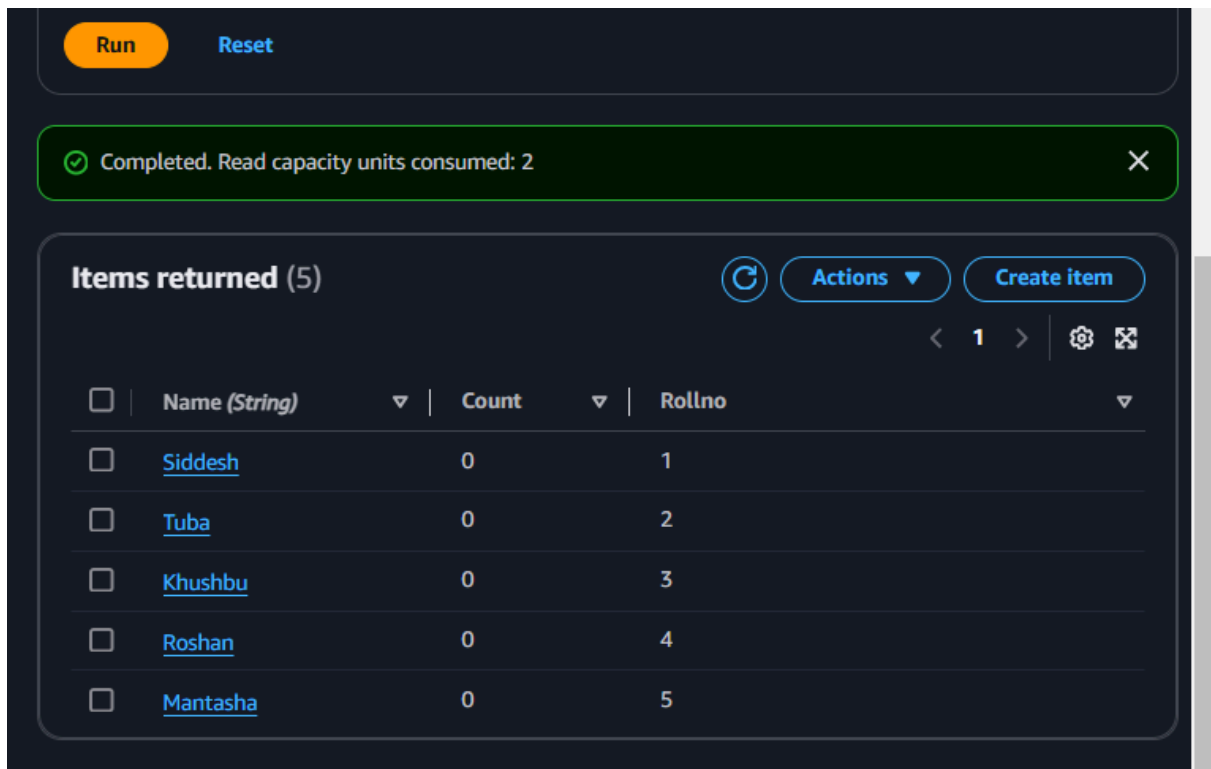
9

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

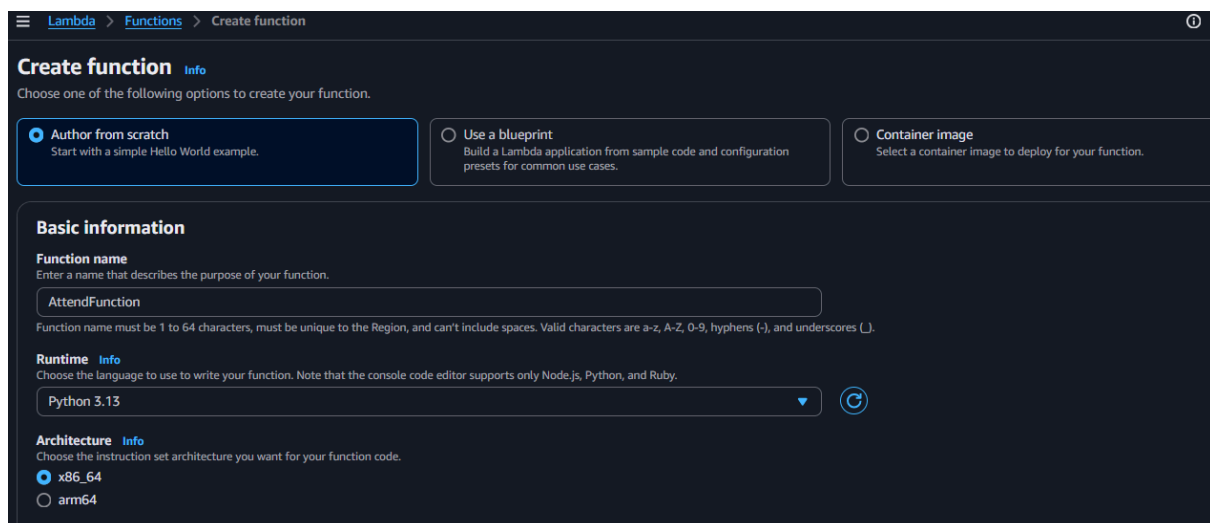
The screenshot shows the AWS Management Console for DynamoDB. The left sidebar contains navigation options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main content area shows the 'AttendTable' configuration page. The 'Tables (1)' list on the left includes 'AttendTable'. The 'AttendTable' configuration page has tabs for Overview, Indexes, Monitor, Global tables, Backups, and Exports and imports. The 'Explore items' tab is selected, showing the 'Scan or query items' section. The 'Scan or query items' section has a 'Scan' button and a 'Query' button. The 'Select a table or index' dropdown shows 'Table - AttendTable'. The 'Select attribute projection' dropdown shows 'All attributes'. The 'Filters' section is empty. The 'Run' button is highlighted. A green status bar at the bottom indicates 'Completed. Read capacity units consumed: 2'.

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

Trigger configuration Info

API Gateway
aws api application-services backend HTTP REST serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports REST, HTTP, and WebSocket APIs. [Learn more](#)

Intent
Use an existing api or have us create one for you.

☒ Create a new API
☐ Use existing API

API type

☐ HTTP API
Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

☐ WebSocket API
Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

☒ REST API
Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Security
Configure the security mechanism for your API endpoint.

Open

► **Additional settings**

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

Code Test Monitor **Configuration** Aliases Versions

General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases
Monitoring and operations tools

Execution role ⌂ Edit View role document

Role name
AttendFunction-role-1nbvsv08 [⌂](#)

Resource summary
To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs
3 actions, 2 resources

By action **By resource**

Resource	Actions
arn:aws:logs:ap-south-1:509399629406:*	Allow: logs:CreateLogGroup

Identity and Access Management (IAM) <

Search IAM

Dashboard
▼ **Access management**
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
▼ **Access reports**
Access Analyzer
External access

Roles > AttendFunction-role-1nbvsv08 ⓘ ⌂

Ⓢ Policies have been successfully attached to role. ×

Last activity - Maximum session duration 1 hour

Permissions Trust relationships Tags Last Accessed Revoke sessions

Permissions policies (3) Info ⌂ Simulate ⌂ Remove Add permissions

You can attach up to 10 managed policies.

Filter by Type All types

Search

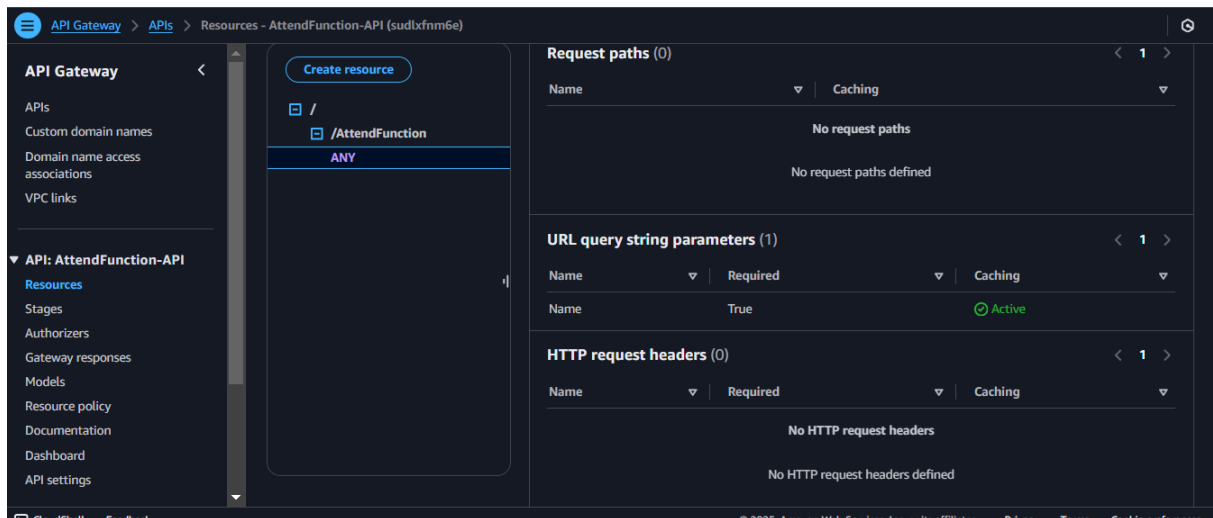
<input type="checkbox"/>	Policy name ⌂	Type	Attached entities
<input type="checkbox"/>	AdministratorAccess	AWS managed - job function	3
<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	1
<input type="checkbox"/>	AWSLambdaBasicExecutionRole-93...	Customer managed	1

Here add dynamodbfull access in the permissions

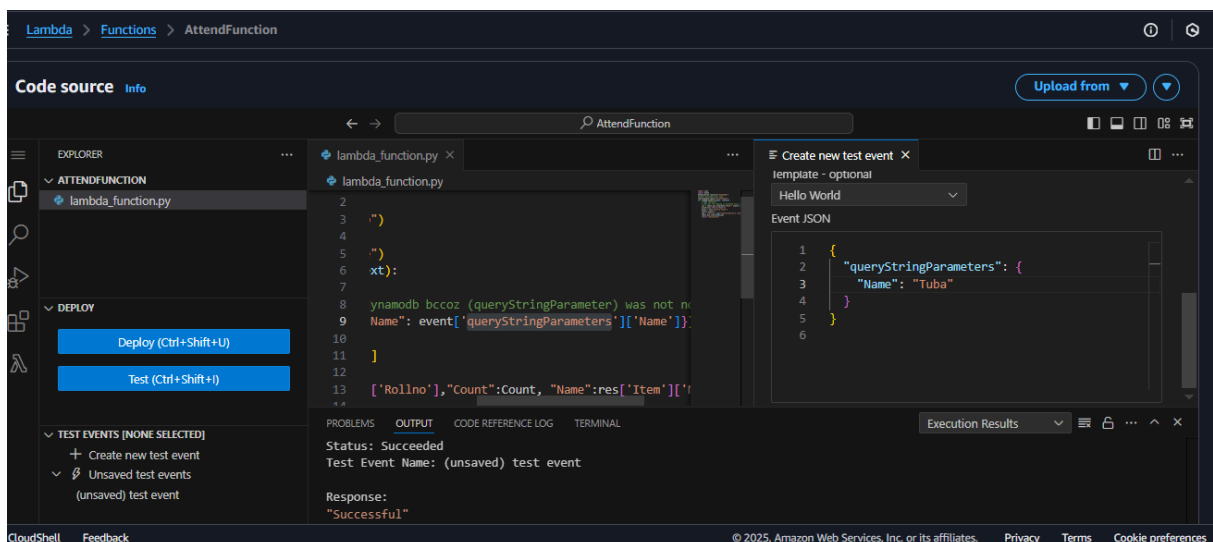
Now give the lambda function python code in the lambda code

Then go to api gateway url from configuration -> triggers

Here add the querystringparameters 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



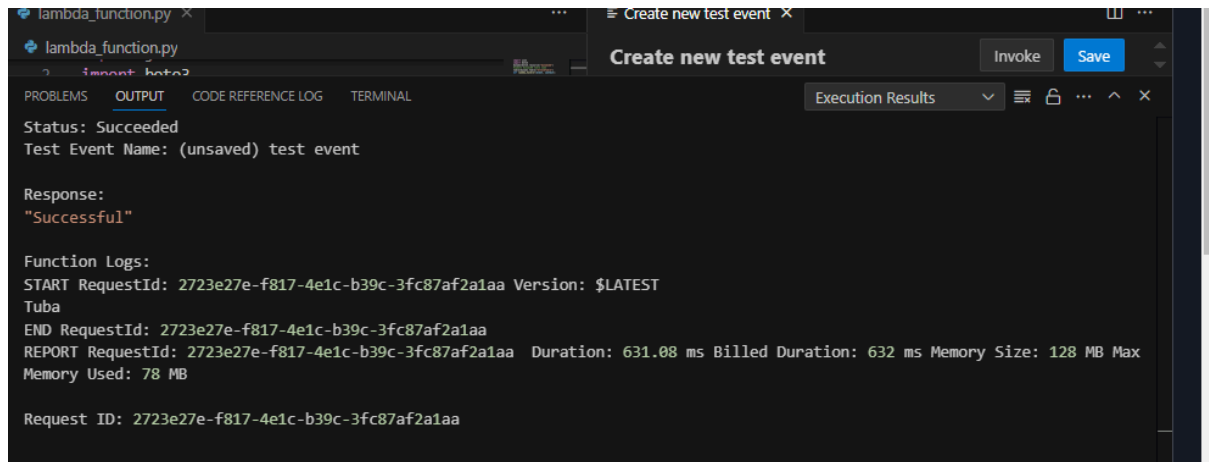
You must see the succeeded status and this will update your table attributes counts as you have given the test code(json)

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

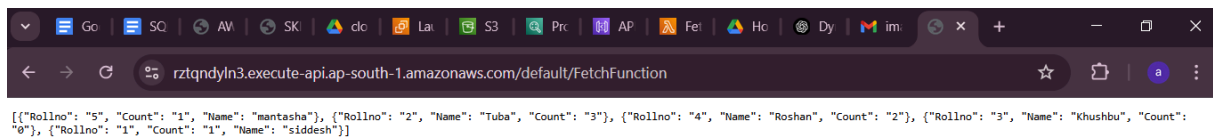
12

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



13

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 (queryStringParameters) 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"])}

```

14

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()

```

15

```

# 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="AKIAXQKH4EDNZY15VFGW",
    aws_secret_access_key="YA4TX/I9LLgR7fwuJnlgZq1ZHga+tWo+M/Fu6Dhl",
    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="YA4TX/l9LLgR7fwuJnlqZq1ZHga+tWo+M/Fu6Dhl",
    region_name='us-east-1')

```

16

```

# 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 :

17

```

JS script.js > document.addEventListener('DOMContentLoaded') callback
1 document.addEventListener('DOMContentLoaded', async () => {
2   try {
3     const response = await fetch('https://rztqndyln3.execute-api.ap-south-1.amazonaws.com/default/Fetc
4     if (!response.ok) {
5       throw new Error('HTTP error! Status: ${response.status}');
6     }
7
8     const data = await response.json();
9     console.log("Fetched Data:", data); // Debugging: Check API response in console
10
11     const tableBody = document.querySelector('#AttendTable tbody');
12     if (!tableBody) {
13       console.error("Error: Table body not found! Make sure you have a <tbody> in your HTML table.")
14     }
15   } catch (error) {
16     console.error(error);
17   }
18 }

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
e': 'application/x-amz-json-1.1', 'content-length': '66', 'date': 'Thu, 20 Feb 2025 01:11:51 GMT', 'RetryAttempts': 0}}
Tuba
Attendance Mark Successful
Exiting script...
PS C:\Users\Ansari Mantasha\Desktop\face_reck> python .\app.py
20-02-25 06-42-17
Image captured!
Recognition Service
{'CustomLabels': [{'Name': 'Roshan', 'Confidence': 74.55699920654297}], 'ResponseMetadata': {'RequestId': '024498cb-350c-4d73-ba
5c-c65aea95a428', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '024498cb-350c-4d73-ba5c-c65aea95a428', 'content-ty
pe': 'application/x-amz-json-1.1', 'content-length': '67', 'date': 'Thu, 20 Feb 2025 01:12:39 GMT', 'RetryAttempts': 0}}
Roshan
Attendance Mark Successful
Exiting script...
PS C:\Users\Ansari Mantasha\Desktop\face_reck>

```

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.textContent = 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/> -

