

"Face Recognition-Based Attendance Management System"

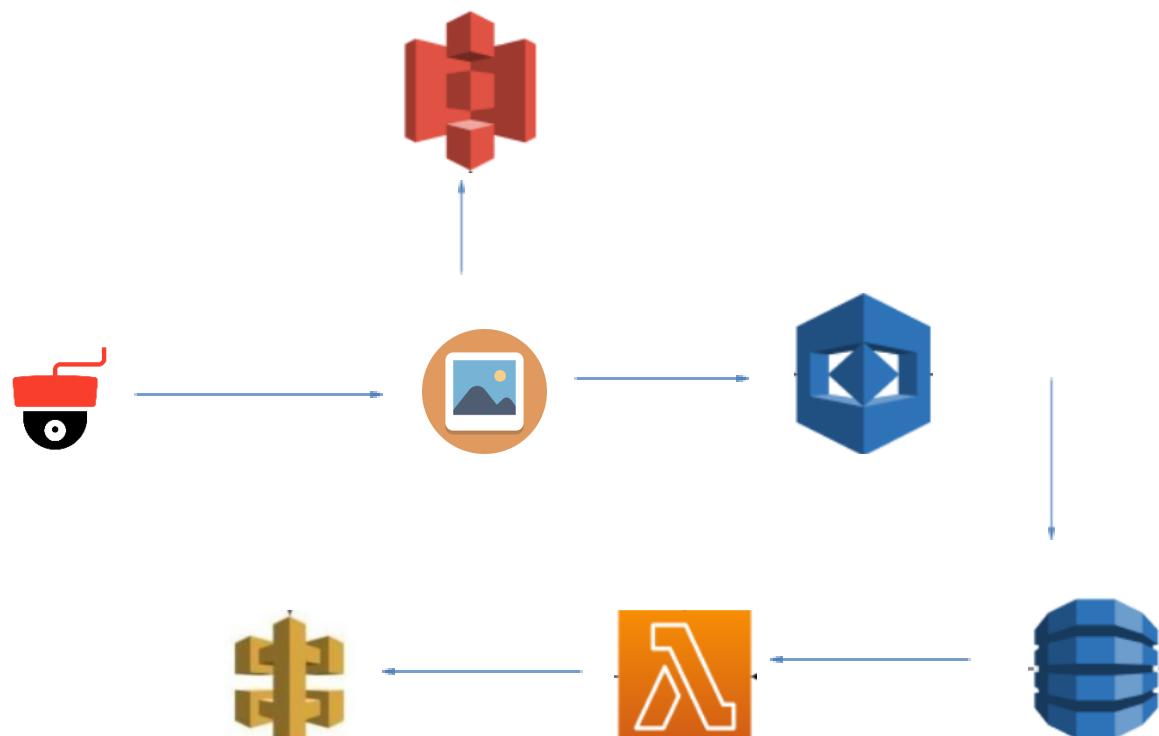
Group Project by:

Ahmed
Mantasha
Tehseen
Ritu

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.

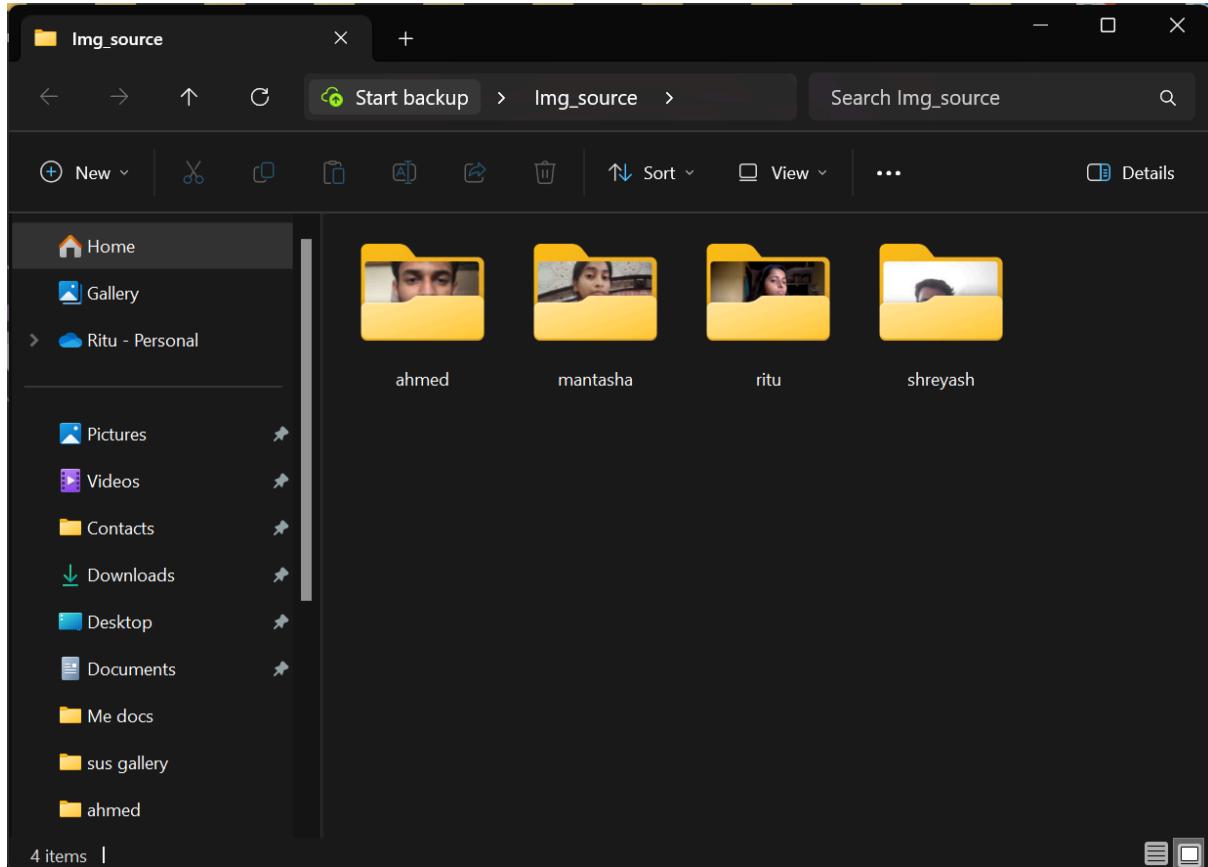
BLOCK DIAGRAM



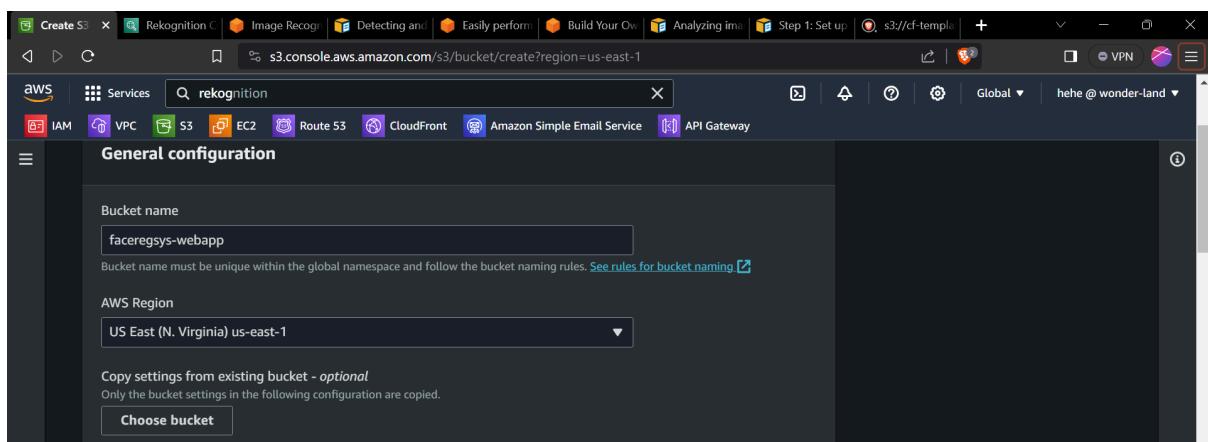
IMPLEMENTATION

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.



Amazon S3 > Buckets > facerec-sys-webapp

facerec-sys-webapp Info

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Actions		Copy S3 URI	Copy URL	Download	Open	Delete	Actions ▾	Create folder	Upload	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="text"/> Find objects by prefix		<input type="checkbox"/> Show versions								
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class					
<input type="checkbox"/>	Img_source/	Folder	-	-	-					

Img_source/

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Actions		Copy S3 URI	Copy URL	Download	Open	Delete	Actions ▾	Create folder	Upload	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="text"/> Find objects by prefix		<input type="checkbox"/> Show versions								
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class					
<input type="checkbox"/>	ahmed/	Folder	-	-	-					
<input type="checkbox"/>	mantasha/	Folder	-	-	-					
<input type="checkbox"/>	ritu/	Folder	-	-	-					
<input type="checkbox"/>	shreyash/	Folder	-	-	-					

- 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."

First time set up

Amazon Rekognition requires an S3 bucket to store your project files.

By continuing, you are giving Amazon Rekognition permission to create an S3 bucket in your account, to store your project files. This is required in order to use the Custom Labels console. [Learn more](#)

S3 bucket name:

Save this location for future reference
custom-labels-console-us-east-1-3c091af861

Create S3 bucket

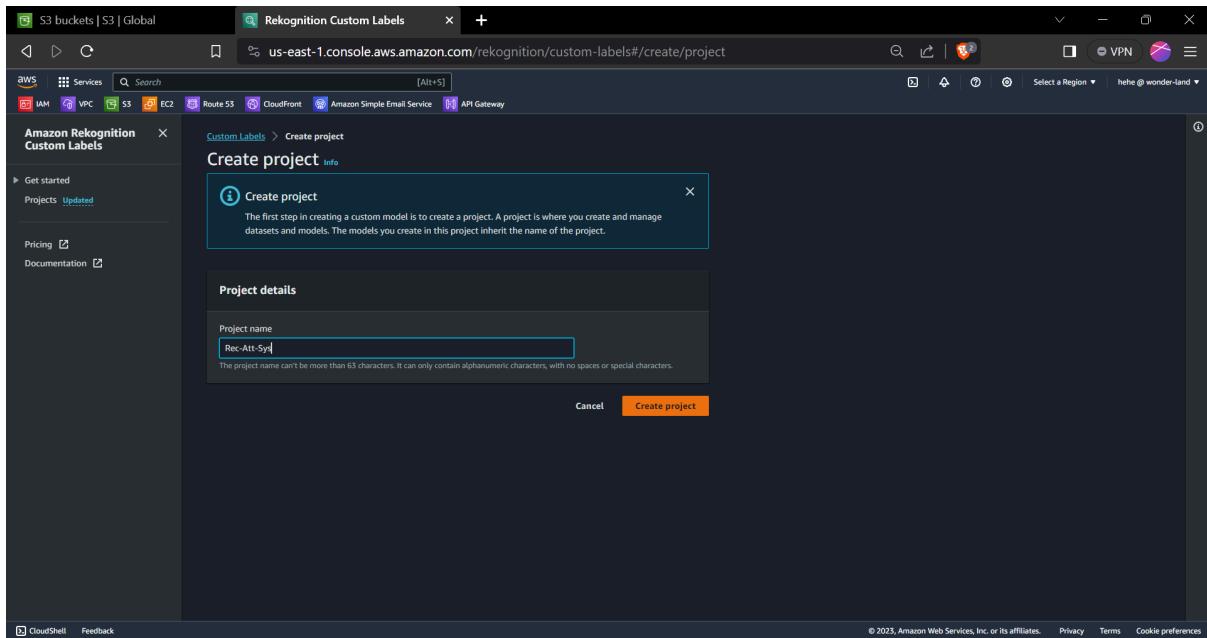
- For the initial setup, you'll be prompted to generate an S3 bucket with a specific name, like '**custom-labels-console-us-east-1-73d5fda5c8**.'
- Recognition functionality is contingent upon existence of this dedicated S3 bucket. Process by selecting "Create S3 bucket."

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)
			October 28, 2023,

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

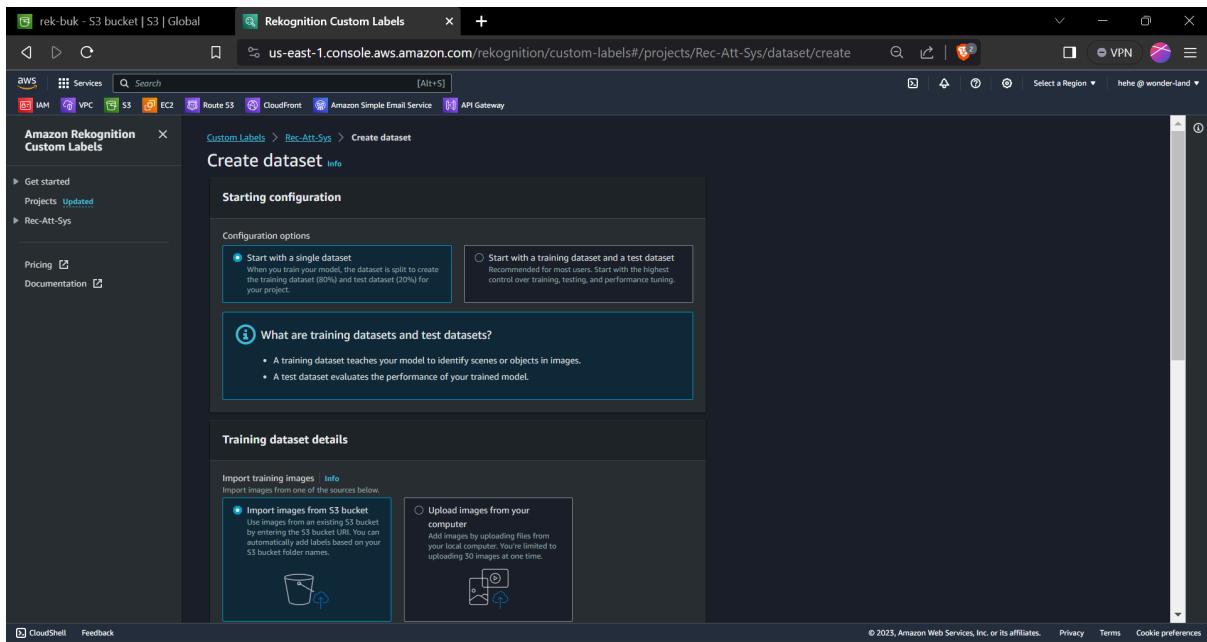
The screenshot shows the AWS Rekognition Custom Labels console. On the left, there's a sidebar with 'Amazon Rekognition Custom Labels' and links for 'Get started' and 'Projects'. The main area is titled 'Projects' and shows a message: 'Projects now manage datasets. Your previous datasets need to be associated with a project. We have associated the previously created datasets that trained the latest version of your models with their corresponding projects. Access your previously created datasets from the Prior datasets node. Learn more'. Below this, there's a search bar and a table header with columns: Name, Versions, Date created, Model performance, Model status, and Status message. A large orange 'Create project' button is prominently displayed at the bottom of the table area.

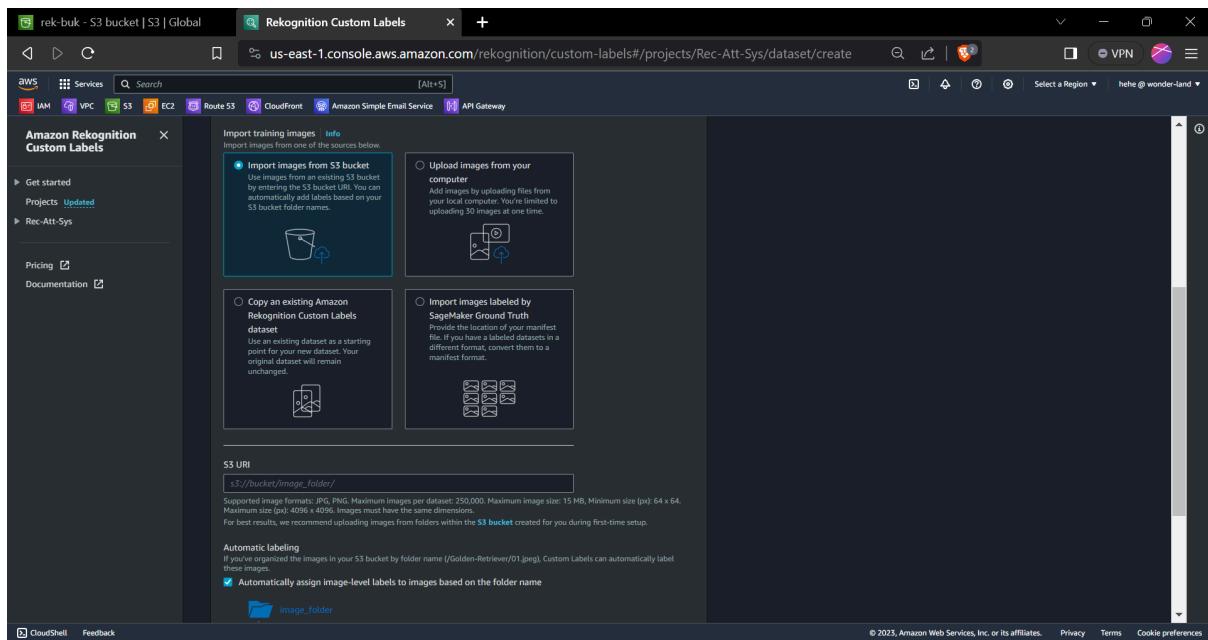
Provide a Project Name of your choice.



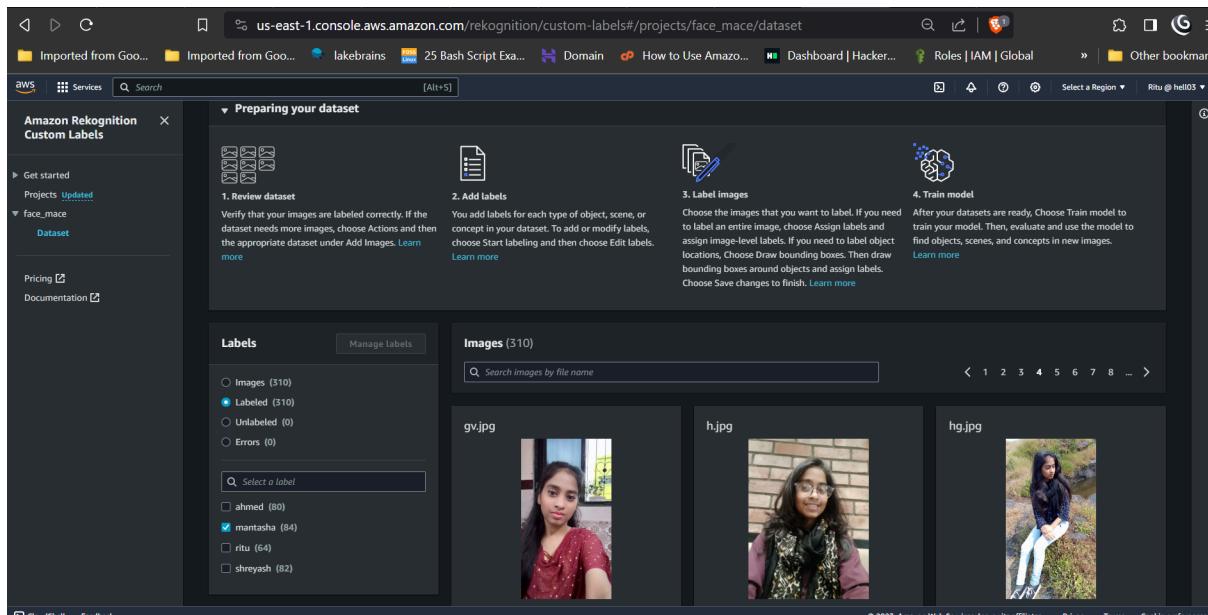
Select "Create Dataset" —> Opt for "Start with Single Dataset" —> Choose "Import images from S3" —> Input the URI of the S3 bucket (The bucket created which we have uploaded data in).

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

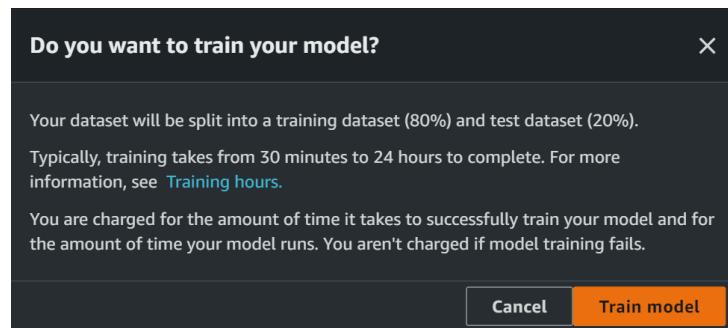
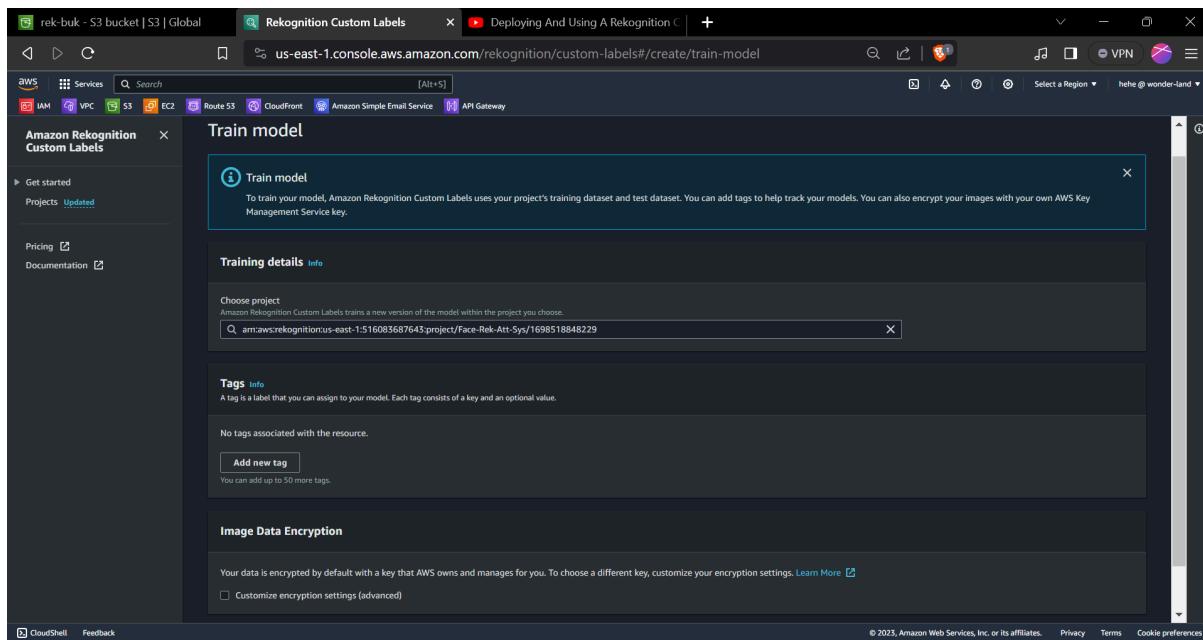




Initiate the labelling process by selecting "Start Labelling," pick the photos for labelling, opt for the "Draw Bounding Boxes" method, and complete the process by clicking "Done."



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

Models (1)					
<input type="button" value="Delete model"/> <input type="button" value="Download validation results"/>					
<input type="text"/> Find resources					
Name	Date created	Training dataset	Test dataset	Model performance (F1 score)	Model status
Face-Rek-Att-Sys.2023-10-29T00.19.42	October 29, 2023		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

Initially, we will establish a table in DynamoDB.

The screenshot shows the 'Create table' wizard in the AWS Management Console. The 'Table details' step is active, showing fields for 'Table name' (set to 'AttendTable'), 'Partition key' (set to 'Name'), and 'Sort key' (optional). The 'Table settings' step is partially visible below. The browser address bar shows the URL: `https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#create-table`. The AWS navigation bar at the top includes services like EC2, VPC, S3, Lambda, and DynamoDB.

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 'AttendTable' details page in the AWS Management Console. The left sidebar shows the 'Tables' section. The main panel displays the 'AttendTable' configuration, including 'General information' (partition key 'Name (String)', sort key '-' (String), capacity mode 'Provisioned', table status 'Active'), 'Items summary' (item count, table size, average item size), and a note about Point-in-time recovery (PITR). The browser address bar shows the URL: `https://us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#item-explore?maximize=true&table=AttendTable`.

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

The screenshot shows the AWS Lambda console. In the left sidebar, under 'Configuration', the 'Execution role' tab is selected. The role name is 'AttendFunction-role-3pv25ru3'. The 'Resource summary' section shows that the function has permissions to access Amazon CloudWatch Logs, specifically allowing actions like 'CreateLogGroup', 'CreateLogStream', and 'PutLogEvents'. The sidebar also lists other configuration options like Triggers, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, Database proxies, and File systems.

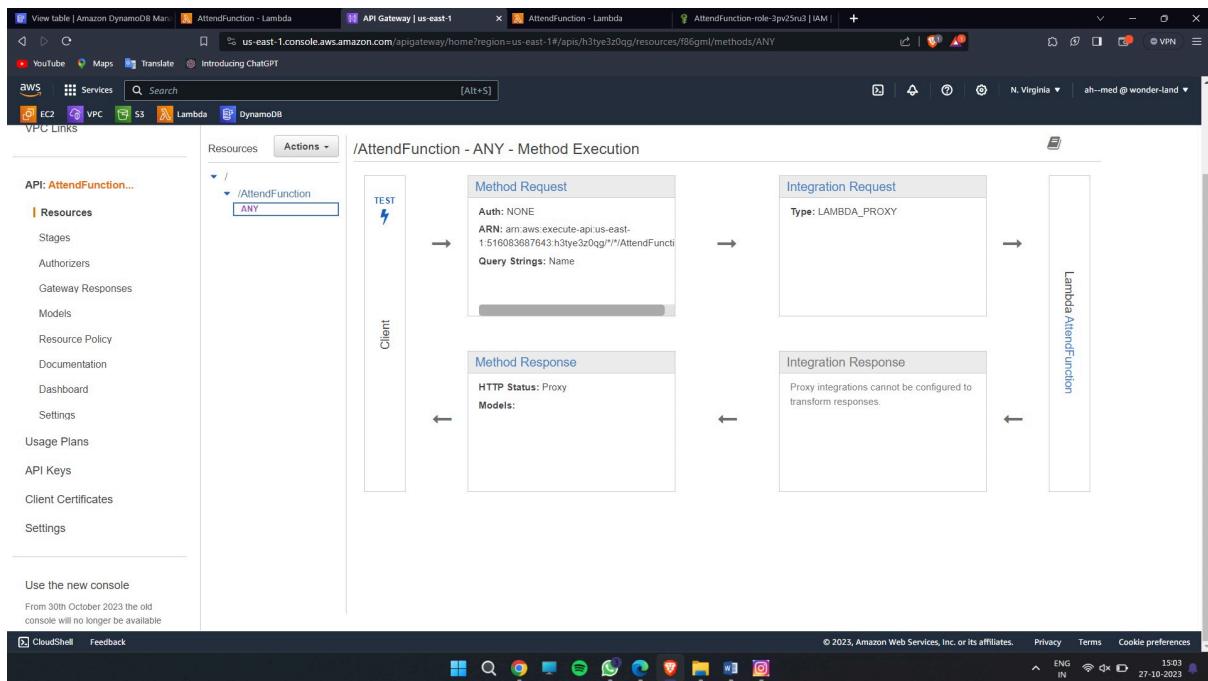
The screenshot shows the AWS IAM Roles details page for the role 'AttendFunction-role-3pv25ru3'. It displays a success message: 'Policies have been successfully attached to role.' The attached policy is 'arn:aws:iam::516083687643:role/service-role/AttendFunction-role-3pv25ru3'. The 'Permissions' tab is selected, showing three managed policies: 'AdministratorAccess', 'AmazonDynamoDBFullAccess', and 'AWSLambdaBasicExecutionRole'. There is also a 'Permissions boundary' section which is currently not set. The left sidebar shows the IAM navigation menu.

Here add dynamodbfull access in the permissions

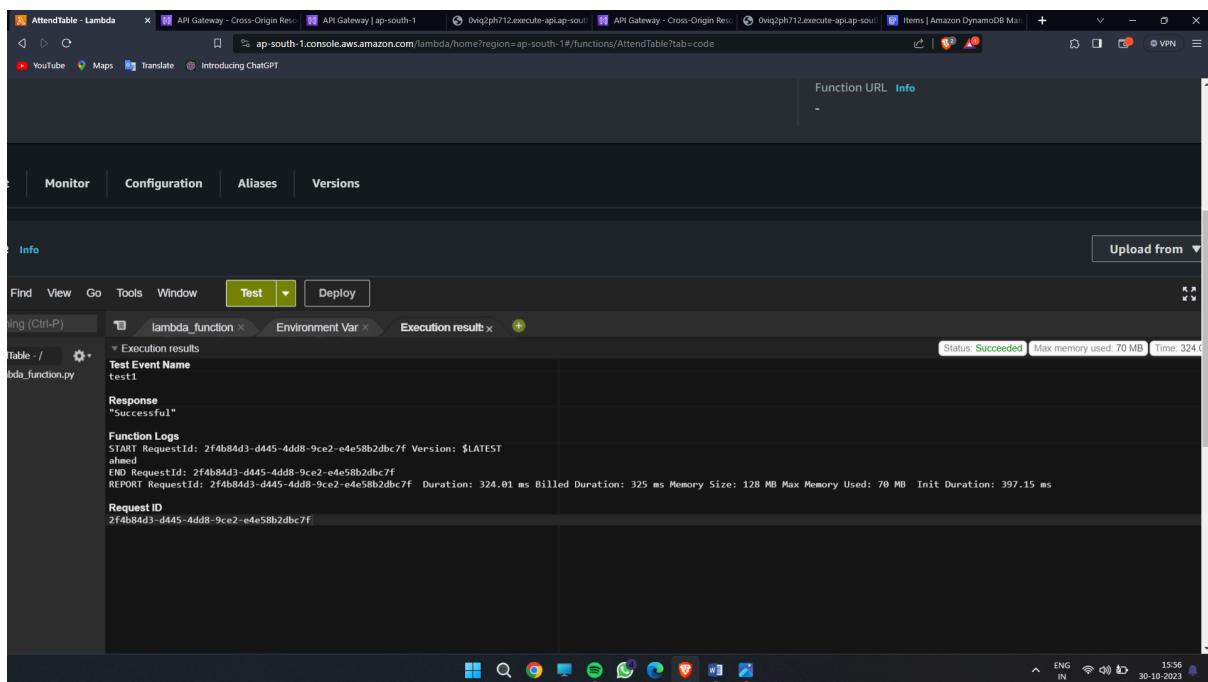
Now give the lambda function python code in the lambda code

Then go to api gateway url from configuration -> triggers

Here add the querystringparamters in the method and deploy the api



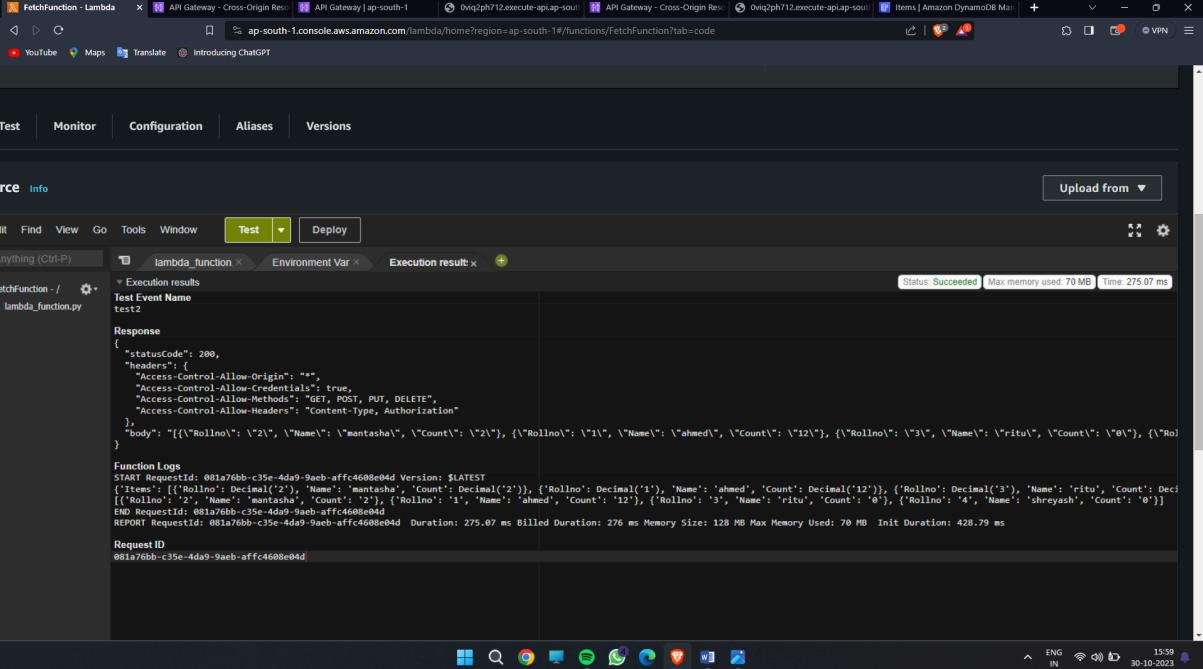
Now you can test the code by creating an event



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

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.

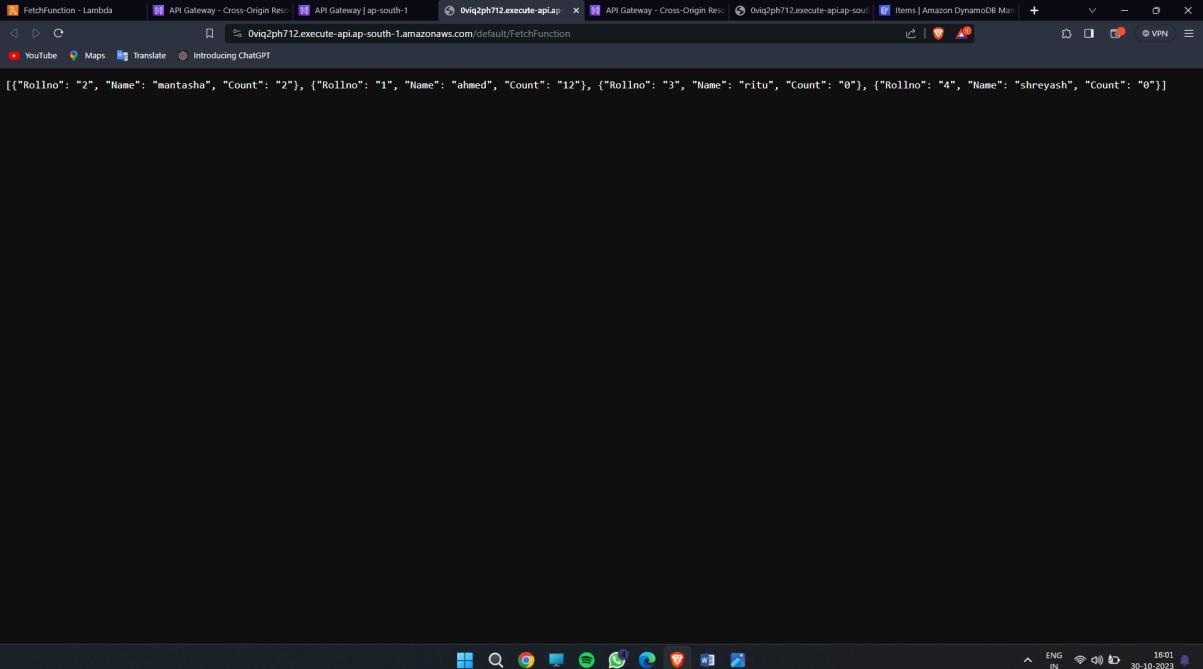


```

{
    "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": "[{"Rollno": "2", "Name": "mantasha", "Count": "2"}, {"Rollno": "1", "Name": "ahmed", "Count": "12"}, {"Rollno": "3", "Name": "ritu", "Count": "0"}, {"Rollno": "4", "Name": "shreyash", "Count": "0"}]"
}

```

Here you should see something like this



```

[{"Rollno": "2", "Name": "mantasha", "Count": "2"}, {"Rollno": "1", "Name": "ahmed", "Count": "12"}, {"Rollno": "3", "Name": "ritu", "Count": "0"}, {"Rollno": "4", "Name": "shreyash", "Count": "0"}]

```

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 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="AKIAQKH4EDNZYI5VFGW",
    aws_secret_access_key="YA4TX/l9LgR7fwuJnlgZq1ZHga+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="AKIAQKH4EDNZYI5VFGW",
    aws_secret_access_key="YA4TX/l9LgR7fwuJnlgZq1ZHga+tWo+M/Fu6Dhl",
    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 :



```
Run attendance x
30-10-23 00-19-57
Image captured!
<class 'bytes'>
Recognition Service
{'CustomLabels': [{'Name': 'mantasha', 'Confidence': 30.30500030517578, 'Geometry': {'BoundingBox': {'Width': 0.3363099992275238, 'Height': 0.5922200083732605, 'Left': 0.353339999141693, 'Top': 0.0}}}], 'ResponseMetadata': {'RequestId': 'a3c3a4b0-4508-4682-96b0-494797238bb2', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'a3c3a4b0-4508-4682-96b0-494797238bb2', 'content-type': 'application/x-amz-json-1.1', 'content-length': '189', 'date': 'Sun, 29 Oct 2023 18:50:07 GMT'}, 'RetryAttempts': 0}}, mantasha
Attendance Mark Sucessful
```

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

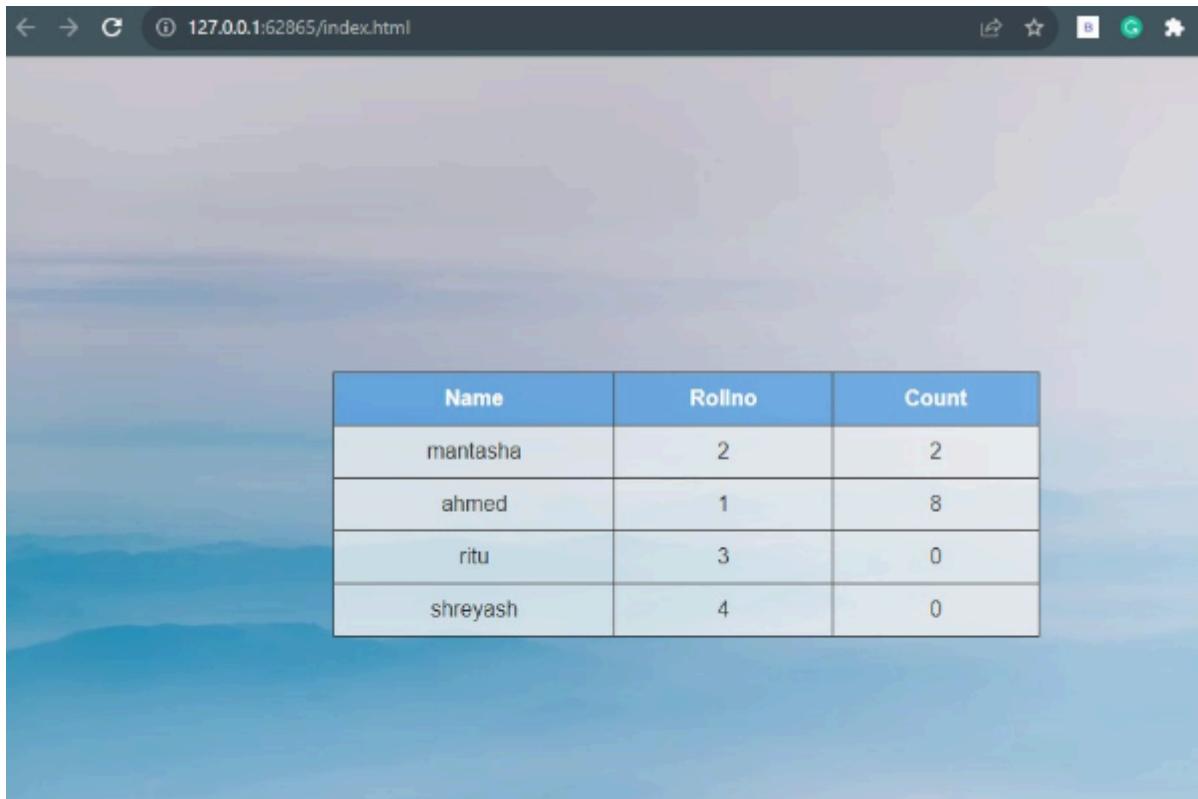
Here we have written code to fetch the data from API and display it in our webpage

```

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:



The screenshot shows a web browser window with the URL 127.0.0.1:62865/index.html. The page displays an attendance table with four rows. The table has three columns: Name, Rollno, and Count. The data is as follows:

Name	Rollno	Count
mantasha	2	2
ahmed	1	8
ritu	3	0
shreyash	4	0

REFERENCE

- <https://github.com/CriMenio/Face-Detection-Attendance-Management-System>
- <https://aws.amazon.com/rekognition/custom-labels-features/>
-