

# **SMART ATTENDANCE USING FACE REKOGNITION**

Prepared in the partial fulfilment of the Summer Internship Program on  
AWS

AT



*Under the guidance of*






***Mrs.Sumana Bethala , APSSDC***





***Mr.AnilKumar, APSSDC***






# SMART ATTENDANCE SYSTEM






## FACE RECOGNITION BASED

### TEAM MEMBERS:

 SUBMITTED BY: TALLAPRAGADA VARSHITHA  
 ROLL NUMBER: 23P31A0512  
 COURSE : AWS CLOUD COMPUTING  
 COLLEGE : ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY  
 BATCH : 2 (SUMANA MAM, ANILKUMAR SIR)

 SUBMITTED BY: KANURI BALA BHAGYA SRI  
 ROLL NUMBER: 23P31A0525  
 COURSE : AWS CLOUD COMPUTING  
 COLLEGE : ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY  
BATCH :2 (SUMANA MAM, ANILKUMAR SIR)

 SUBMITTED BY: BARNINKALA RAMYA  
 ROLL NUMBER: 23P31A05A0  
 COURSE : AWS CLOUD COMPUTING  
 COLLEGE : ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY  
 BATCH :2 (SUMANA MAM, ANILKUMAR SIR)

 SUBMITTED BY: ARUGULA SANTHOSH KUMAR  
 ROLL NUMBER: 23P31A0580  
 COURSE : AWS CLOUD COMPUTING  
 COLLEGE : ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY  
 BATCH :2 (SUMANA MAM, ANILKUMAR SIR)

## **ABSTRACT:**

This project presents a Smart Attendance System that uses facial recognition to automate the traditional attendance process. The system utilizes a web application where students can register their face images.

During attendance, live images are captured and compared using Amazon Rekognition. Images and logs are securely stored in Amazon S3, with serverless logic handled by AWS Lambda.

The system ensures accuracy, prevents proxy attendance, and provides realtime access to data via an admin dashboard.

## **INTRODUCTION:**

Problem Statement: Manual attendance systems are time-consuming, error prone, and often vulnerable to proxies or manipulation.

Objective:

To develop a smart, cloud-based facial recognition attendance system that automates attendance, ensures data security, and prevents proxy attendance.

## **Technology Stack Used:**

The Smart Attendance System leverages a full AWS serverless architecture integrated with modern frontend technologies to provide a scalable and efficient attendance management solution using face recognition.

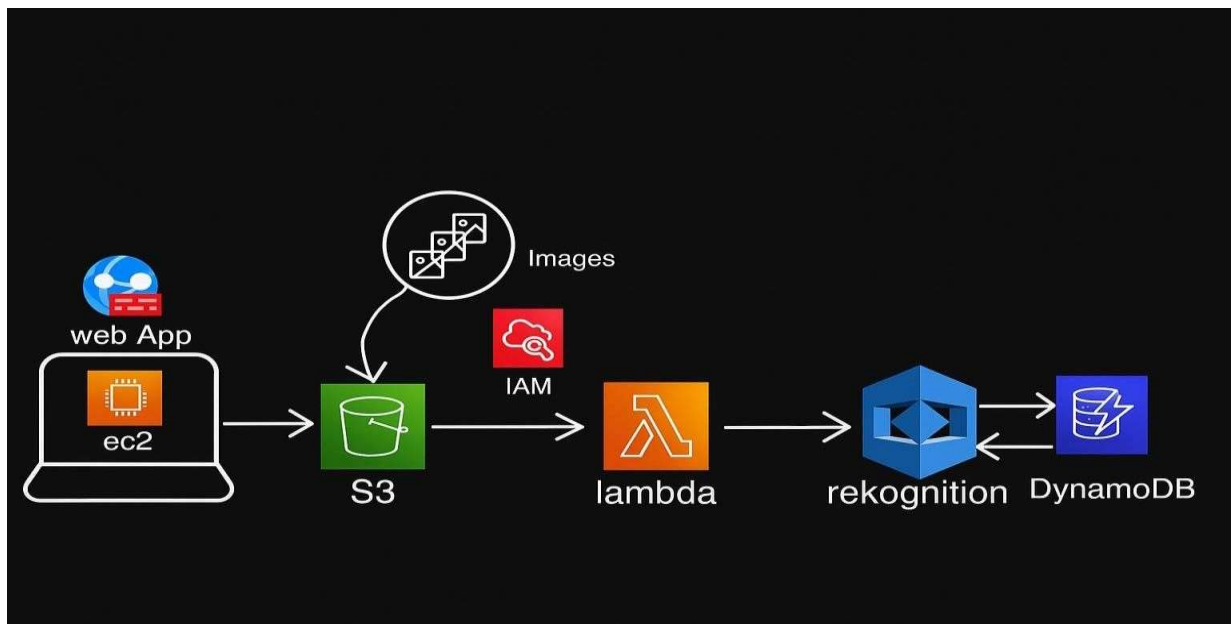
- Backend:
  - AWS Lambda (Python/Node.js) – Serverless compute service to handle backend logic such as processing images, triggering face detection, and updating attendance records.

- Face Detection:

Amazon Rekognition – A powerful AI service used for facial recognition. It compares uploaded images with stored faces to identify students and mark attendance.

- Storage:
  - Amazon S3 – Stores student face images and attendance logs in organized buckets, ensuring high availability and durability.
- API Gateway:
  - Amazon API Gateway – Acts as a front door for the backend APIs, allowing secure and scalable communication between the frontend and AWS Lambda functions.
- Database:
  - Amazon DynamoDB – A NoSQL database used to store structured student data, attendance records, and metadata for quick retrieval and scalability.
- Hosting:
  - Amazon S3 Static Website Hosting – Hosts the frontend web application securely and efficiently.

## System Design / Architecture:



### System Components:

- API Gateway: Connects frontend to backend services.

- AWS Lambda: Executes core logic and image processing.
- Amazon Rekognition: Performs facial recognition to identify faces.
- Amazon S3: Stores images and attendance logs.
- DynamoDB / RDS (Optional): Stores structured records like student data and attendance history.

Optional Visual:

Add a simple architecture diagram (can be AWS-style) showing:

- Web App → API Gateway → Lambda
- Lambda ↔ Rekognition
- Lambda ↔ S3 & DynamoDB

## **Implementation:**

Welcome to our step-by-step process on creating a Facial Recognition System powered by AWS Rekognition.

Two main modules:

Face Registration: Uploads image to S3 and indexes with Rekognition.

Marking of Faces: Captures webcam image, compares with Rekognition, and logs attendance.

## **Step 1: Setting Up Infrastructure**

- Creating EC2 Instance: We'll start by launching an EC2 instance to serve as our computing environment.
- Configuring AWS CLI: Learn how to configure the AWS Command Line Interface for seamless interaction with AWS services.
- Creating AWS Rekognition Collection: We'll create a Rekognition collection to store and manage face data effectively.

- Cloning GitHub Repository: Get hands-on with cloning a GitHub repository containing essential resources for our project.

## Step 2: Configuring Core Components

- Creating S3 Bucket: Set up an S3 bucket to store images and other resources.
- Creating DynamoDB Table: Learn how to create a DynamoDB table to store metadata and other relevant information.
- Creating Lambda Function: Discover the process of creating a Lambda function and configuring an S3 bucket as a trigger.

## Step 3: Implementing Facial Recognition

- Uploading Images to S3: Upload images to the S3 bucket, triggering the Lambda function to generate face prints.
- Generating Face Prints: Witness the Lambda function in action as it generates face prints of uploaded images.
- Storing Face Prints in DynamoDB: Explore how face prints are uploaded to the DynamoDB table for efficient storage and retrieval.
- Running Web Application with Docker: Finally, we'll deploy a web application using Docker, showcasing the integration of our Facial Recognition System into a user-friendly interface.

## **Procedure:**

Launch EC2 instance:

aws Search [Alt+S] Europe (Stockholm) Santhosh Kumar Arugula

EC2 > Instances > Launch an instance

**Success**  
Successfully initiated launch of instance (i-0477968bd68c3da03)

► Launch log

**Next Steps**  
What would you like to do next with this instance, for example "create alarm" or "create backup"

**Create billing and free tier usage alerts**  
To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds.  
[Create billing alerts](#)

**Connect to your instance**  
Once your instance is running, log into it from your local computer.  
[Connect to instance](#)  
[Learn more](#)

**Connect an RDS database**  
Configure the connection between an EC2 instance and a database to allow traffic flow between them.  
[Connect an RDS database](#)  
[Create a new RDS database](#)  
[Learn more](#)

**Create EBS snapshot policy**  
Create a policy that automates the creation, retention, and deletion of EBS snapshots.  
[Create EBS snapshot policy](#)

**Manage detailed monitoring** **Create Load Balancer** **Create AWS budget** **Manage CloudWatch alarms**

## IAM Role:

aws Search [Alt+S] Global Santhosh Kumar Arugula

IAM > Roles

**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  
Resource analysis [New](#)  
Unused access  
Analyzer settings  
Credential report

**Role FaceRecognition\_lambda\_role created.** [View role](#)

**Roles (5)** [info](#) [Delete](#) [Create role](#)

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Search

<input type="checkbox"/>	Role name	Trusted entities	Last activity
<input type="checkbox"/>	<a href="#">AWSServiceRoleForSupport</a>	AWS Service: support ( <a href="#">Service-Linker</a> )	-
<input type="checkbox"/>	<a href="#">AWSServiceRoleForTrustedAdvisor</a>	AWS Service: trustedadvisor ( <a href="#">Service</a> )	-
<input type="checkbox"/>	<a href="#">faceaccesslamdaflow</a>	AWS Service: lambda	2 days ago
<input type="checkbox"/>	<a href="#">FaceRecognition_lambda_role</a>	AWS Service: lambda	-
<input type="checkbox"/>	<a href="#">FacialRecognitionLambdaRole</a>	AWS Service: lambda	19 hours ago

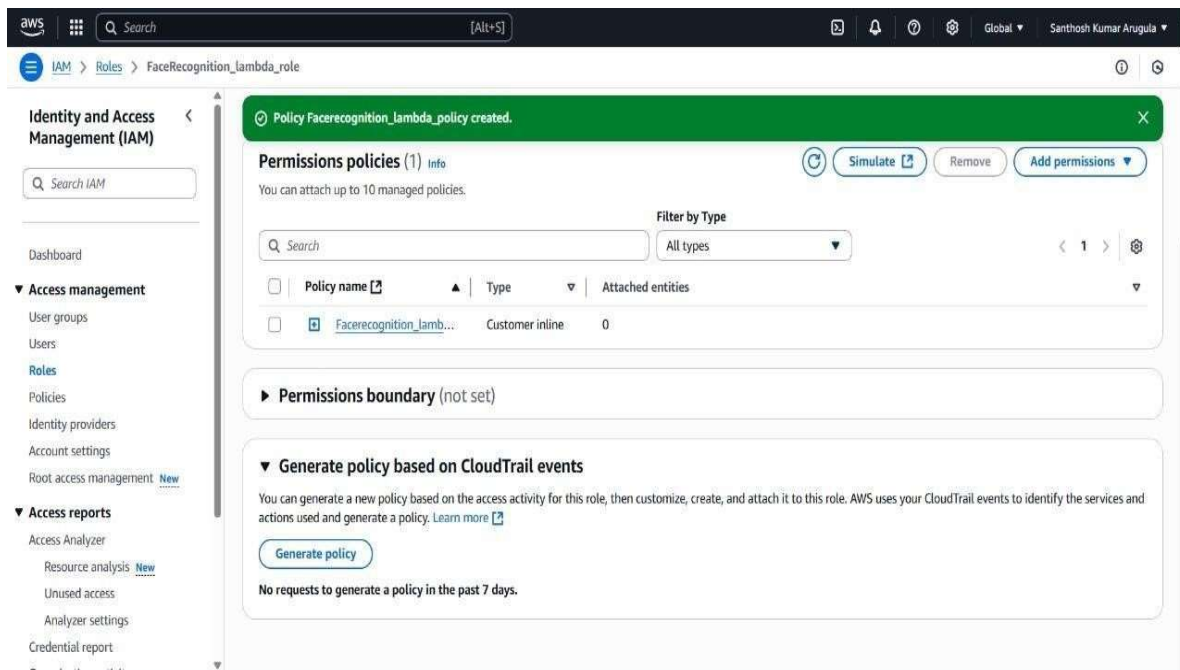
**Roles Anywhere** [info](#) [Manage](#)  
Authenticate your non AWS workloads and securely provide access to AWS services.

**Access AWS from your non AWS workloads**  
Operate your non AWS workloads using the same

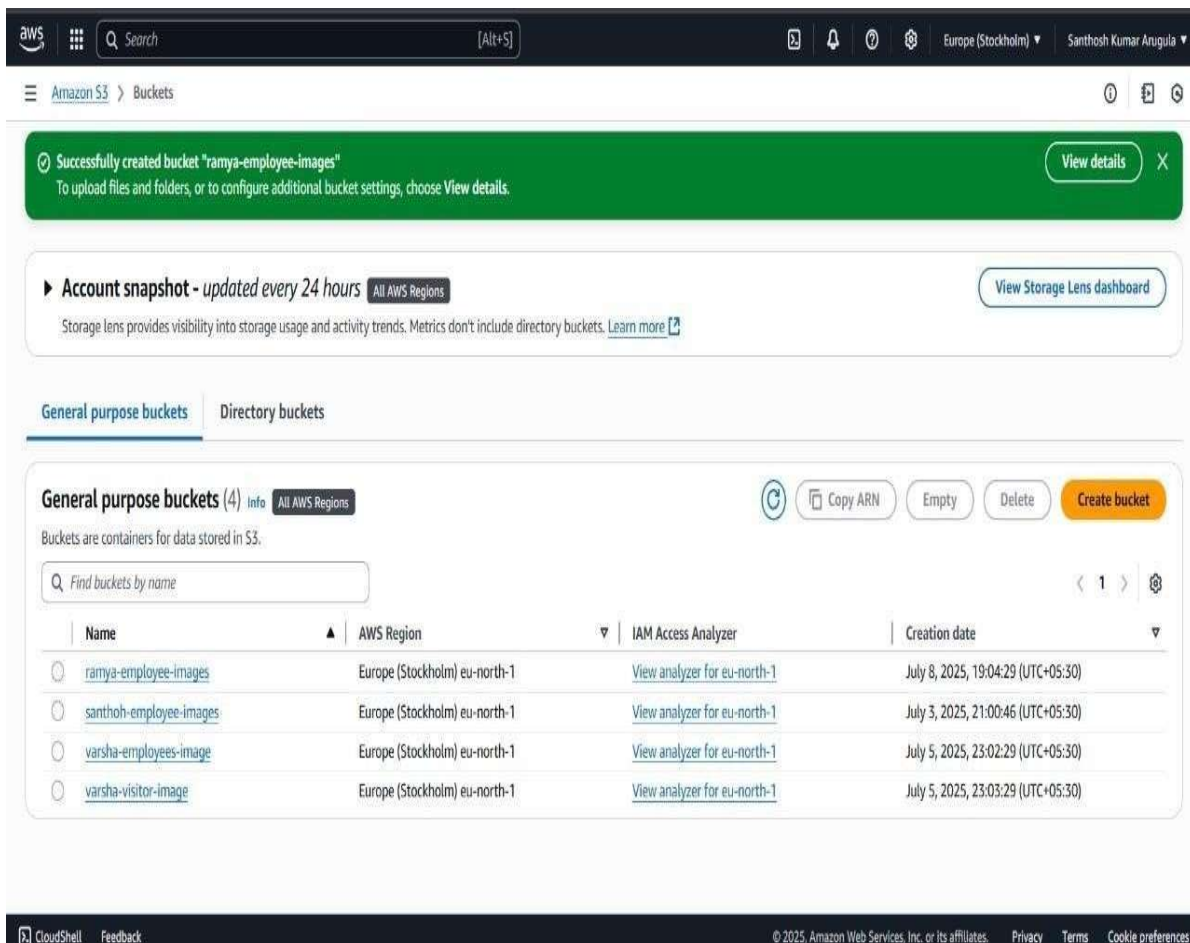
**X.509 Standard**  
Use your own existing PKI infrastructure or use [AWS](#)

**Temporary credentials**  
Use temporary credentials with ease and benefit from

## IAM Policy:



## Create a S3 Bucket:



## Create a Lambda Function:

aws Search [Alt+S] Europe (Stockholm) Santhosh Kumar Arugula

Lambda > Functions > facerecognition\_function

Successfully created the function **facerecognition\_function**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

**facerecognition\_function** Throttle Copy ARN Actions

Function overview info Export to Infrastructure Composer Download

Diagram Template

facerecognition\_function Layers (0)

+ Add trigger + Add destination

Description

Last modified 14 seconds ago

Function ARN [arn:aws:lambda:eu-north-1:089081312175:function:facerecognition\\_function](#)

Function URL Info

Code Test Monitor Configuration Aliases Versions

Code source info Upload from

## Update a Lambda Function:

aws Search [Alt+S] Europe (Stockholm) Santhosh Kumar Arugula

Lambda > Functions > facerecognition\_function

Successfully updated the function **facerecognition\_function**.

EXPLORER

lambda\_function.py l\_function.py X

▼ FACERECOGNITION\_FUNC... l\_function.py

▼ DEPLOY Deploy (Ctrl+Shift+U) Test (Ctrl+Shift+I)

▼ TEST EVENTS (NONE SELEC... + Create new test event

▼ ENVIRONMENT VARIABLES

```

1 import boto3
2 from decimal import Decimal
3 import json
4 import urllib.parse
5
6 print('Loading function')
7
8 dynamodb = boto3.client('dynamodb')
9 s3 = boto3.client('s3')
10 rekognition = boto3.client('rekognition')
11
12 # ----- Helper Functions -----
13
14 def index_faces(bucket, key):
15     response = rekognition.index_faces(
16         Image={'S3Object': {'Bucket': bucket, 'Name': key}},
17         CollectionId='cricketers'
18     )
19     return response
20
21 def update_index(tableName, faceId, fullName):
22     response = dynamodb.put_item(
23         TableName=tableName,
24         Item={
25             'RecognitionId': {'S': faceId},
26             'FullName': {'S': fullName}
27         }
28     )

```

Successfully updated the function facerecognition\_function.

## Create Key Access:

IAM > Users > Students > Create access key

**Access key created**  
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Step 1  
Access key best practices & alternatives

Step 2 - optional  
Set description tag

Step 3  
**Retrieve access keys**

### Retrieve access keys Info

**Access key**  
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key: AKIARJPNIGQXTZEXTLJ

Secret access key: \*\*\*\*\* [Show](#)

**Access key best practices**

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#) [Done](#)

## Creating a Dynamodb Tables:

aws Search [Alt+S]

Europe (Stockholm) Santhosh%20Kumar%20Arugula

DynamoDB > Tables

**DynamoDB**

Dashboard

**Tables**

Explore items

PartiQL editor

Backups

Exports to S3

Imports from S3

Integrations [New](#)

Reserved capacity

Settings

**▼ DAX**

Clusters

Subnet groups

Parameter groups

Events

**Tables (3) Info**

[Find tables](#) Any tag key Any tag value < 1 > ⚙️

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Ri
<input type="checkbox"/>	<a href="#">employee</a>	Active	rekognitionId (S)	-	0	0	Off	☆	0
<input type="checkbox"/>	<a href="#">Employees_collection</a>	Active	Recognitionid (S)	-	0	0	Off	☆	0
<input type="checkbox"/>	<a href="#">face</a>	Active	rekognitionid (S)	-	0	0	Off	☆	0

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

## Installation of Docker to run the Application:

Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1029-aws x86\_64)

\* Documentation: <https://help.ubuntu.com>  
 \* Management: <https://landscape.canonical.com>  
 \* Support: <https://ubuntu.com/pro>

System information as of Sat Jul 12 06:27:24 UTC 2025

System load: 0.08 Temperature: -273.1 C  
 Usage of /: 42.2% of 6.71GB Processes: 107  
 Memory usage: 31% Users logged in: 0  
 Swap usage: 0% IPv4 address for ens5: 172.31.25.115

\* Ubuntu Pro delivers the most comprehensive open source security and compliance features.

<https://ubuntu.com/aws/pro>

Expanded Security Maintenance for Applications is not enabled.

20 updates can be applied immediately.

To see these additional updates run: `apt list --upgradable`

Enable ESM Apps to receive additional future security updates.

See <https://ubuntu.com/esm> or run: `sudo pro status`

\*\*\* System restart required \*\*\*

i-01a1916e1637c2452 (Final\_faceapp)

PublicIPs: 56.228.9.139 PrivateIPs: 172.31.25.115

compliance features.

<https://ubuntu.com/aws/pro>

Expanded Security Maintenance for Applications is not enabled.

20 updates can be applied immediately.

To see these additional updates run: `apt list --upgradable`

Enable ESM Apps to receive additional future security updates.

See <https://ubuntu.com/esm> or run: `sudo pro status`

\*\*\* System restart required \*\*\*

Last login: Fri Jul 11 16:28:43 2025 from 13.48.4.202

ubuntu@ip-172-31-25-115:~\$ sudo apt-get update

Hit:1 <http://eu-north-1.ec2.archive.ubuntu.com/ubuntu> noble InRelease

Get:2 <http://eu-north-1.ec2.archive.ubuntu.com/ubuntu> noble-updates InRelease [126 kB]

Hit:3 <http://eu-north-1.ec2.archive.ubuntu.com/ubuntu> noble-backports InRelease

Hit:4 <http://security.ubuntu.com/ubuntu> noble-security InRelease

Fetched 126 kB in 0s (404 kB/s)

Reading package lists... Done

ubuntu@ip-172-31-25-115:~\$ sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Reading package lists... Done

Building dependency tree... Done

Reading state information... Done

Package docker-ce is not available, but is referred to by another package.

This may mean that the package is missing, has been obsoleted, or

i-01a1916e1637c2452 (Final\_faceapp)

PublicIPs: 56.228.9.139 PrivateIPs: 172.31.25.115

## Python Code:

```
from flask import Flask, render_template, request import
```

```
boto3
```

```

import io      from PIL import Image app = Flask(__name__) rekognition
= boto3.client('rekognition', region_name='us-east-1') dynamodb
= boto3.client('dynamodb', region_name='us-east-1')

@app.route('/', methods=['GET', 'POST'])
def index():    if request.method == 'POST':
image_file = request.files['image_path']
image = Image.open(image_file)    stream
= io.BytesIO()    image.save(stream,
format="JPEG")    image_binary =
stream.getvalue()

    response = rekognition.search_faces_by_image(
        CollectionId='cricketers',
        Image={'Bytes': image_binary}
    )

    found = False    recognized_faces = []
for    match    in    response['FaceMatches']:
face_id = match['Face']['FaceId']    confidence
= match['Face']['Confidence']

    face = dynamodb.get_item(
        TableName='cricketers_collection',
        Key={'RekognitionId': {'S': face_id}}
    )
    if 'Item' in
face:
recognized_faces.append
end(face['Item']['FullName']['S'])

found = True

```

```
        if
found:
    return render_template('result.html', recognized_faces=recognized_faces)
else:
    return render_template('result.html', error="Person cannot be
recognized")

return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

## Results/Output:

- 1)Faces successfully registered.
- 2)Smart Attendance System (Face Recognition Based).
- 3)Real-time attendance marking using webcam.
- 4)Logs stored in S3 as JSON/CSV.

## Conclusion & Future Scope:

The Smart Attendance System successfully automates attendance, enhances accuracy, and ensures data security using AWS cloud services.

With Amazon Rekognition, it provides accurate and contact less facial recognition. AWS Lambda, API Gateway, and S3 create a scalable, serverless backend with low maintenance.

The web interface is user-friendly, enabling real-time registration and monitoring. Data is securely stored in S3, with optional use of DynamoDB or RDS. Optional Cognito integration ensures secure user authentication.

The system reduces manual errors and workload while enhancing transparency.

Overall, it is a robust, efficient, and scalable solution for modern institutions.

## References:

- AWS Documentation: <https://docs.aws.amazon.com>
- Basics Lambda + API Gateway Tutorials
- Amazon Rekognition Guide
- MDN Web Docs: WebRTC & JS