Bachelor of Technology (B. Tech.)

Amrita School of Engineering

Coimbatore Campus (India)

Academic Year – 2023 – 2024

Semester - 7

CLOUD COMPUTING CASE-STUDY DOCUMENT

Name	Roll Number
R Rangashree Dhanvanth	CB.EN.U4CCE20048

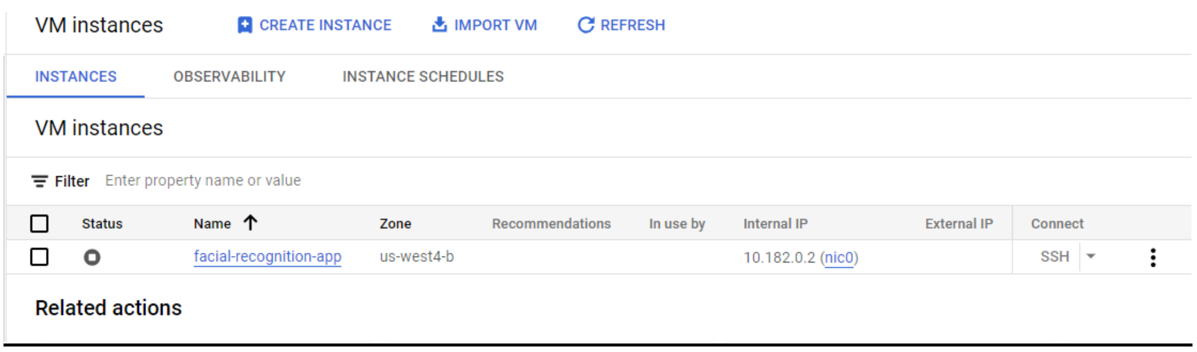
FACE RECOGNITION USING AWS CLOUD AND AMAZON REKOGNITION

ABSTRACT:

This project leverages Amazon Web Services (AWS) to implement a sophisticated face recognition system for office security. Utilizing AWS Rekognition, the system is designed to identify and authenticate individuals accessing the premises. The core functionality is encapsulated within AWS Lambda functions, housing the facial recognition codes for efficient and scalable execution. Data management is facilitated through Amazon DynamoDB, providing a reliable and scalable NoSQL database for storing crucial information. To enhance organization and security, two distinct S3 buckets are employed—one dedicated to visitor data and another for employee records. AWS RestAPI ensures seamless communication and interaction with the system, offering a user-friendly interface for managing access and monitoring activities. This comprehensive AWS-based solution integrates cutting-edge facial recognition technology with robust cloud services to optimize office security and streamline access management.

CLOUD PLATFORMS AND SERVICES USED:

1. **Google cloud platform:** The website is hosted on Google Cloud Platform (GCP) using a Virtual Machine (VM) instance, ensuring reliable and scalable performance. Leveraging GCP's Compute Engine, a VM instance is provisioned to accommodate the web application's requirements. This virtualized environment offers flexibility in resource allocation and seamless integration with other GCP services. The deployment is streamlined, allowing for efficient management and optimization of computing resources. Additionally, GCP's robust networking capabilities ensure low-latency access to the face recognition application, enhancing overall user experience while maintaining the security and reliability expected from a cloud hosting solution.



Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
●	facial-recognition-app	us-west4-b			10.182.0.2 (nic0)		SSH

2. **AWS Rekognition:** AWS Rekognition is employed for visitor face recognition, enhancing office security. Leveraging advanced machine learning algorithms, Rekognition accurately identifies and verifies individuals entering the premises. Amazon Rekognition is a cloud-based image and video analysis service offered by Amazon Web Services (AWS). Designed with advanced machine learning capabilities, Rekognition enables developers to incorporate powerful visual recognition features into their applications without requiring expertise in machine learning.

Key features of AWS Rekognition include:

- **Facial Recognition:** Rekognition can identify and analyze faces in images and videos, allowing for face detection, facial analysis (such as emotion detection), and facial matching.

- **Object and Scene Detection:** The service can identify and categorize objects and scenes within images, providing valuable insights into the content.

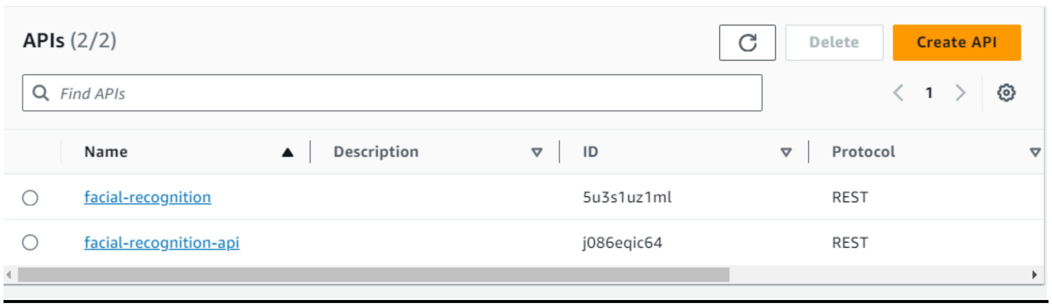
- **Text Recognition:** Rekognition can recognize and extract text from images, making it useful for applications involving document analysis and extraction.

- **Content Moderation:** It includes content moderation capabilities to automatically detect and filter inappropriate content in images and videos.

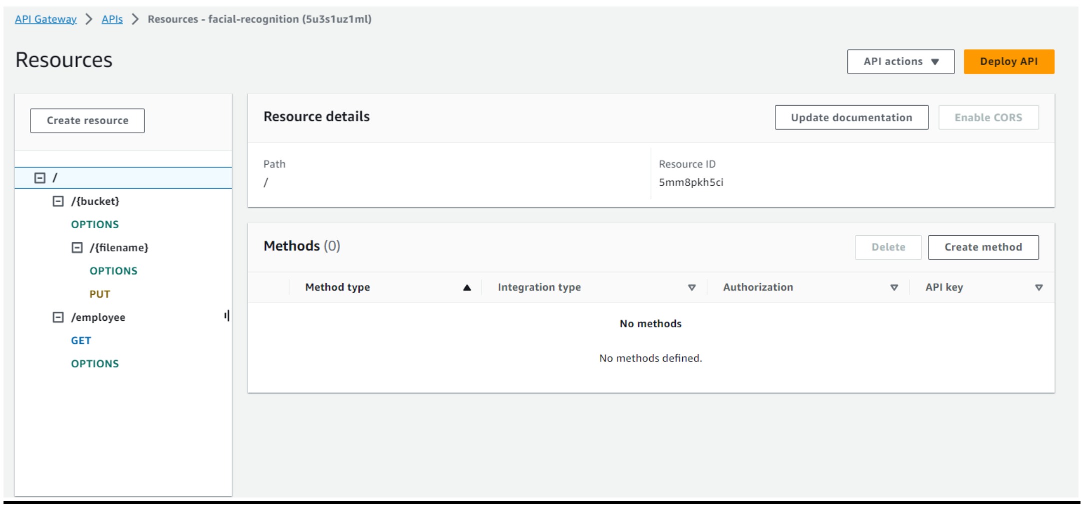
- **Custom Labels:** Users can train Rekognition to recognize specific objects or scenes relevant to their use case by creating custom labels.

- **Celebrity Recognition:** The service can recognize well-known celebrities in images.

3. **API Gateway:** The AWS API Gateway, configured as a RESTful API, serves as a crucial component for accessing the face recognition website. By implementing GET and PUT methods, it enables seamless communication between clients and the underlying AWS services. The GET method facilitates the retrieval of information, allowing users to access data related to face recognition activities. On the other hand, the PUT method supports the submission of new data or updates, ensuring dynamic interaction with the system. This RESTful API, integrated with the website, provides a standardized and secure interface for users to retrieve and update information, enhancing the overall functionality and user experience.



Name	Description	ID	Protocol
facial-recognition		5u3s1uz1ml	REST
facial-recognition-api		j086eqic64	REST



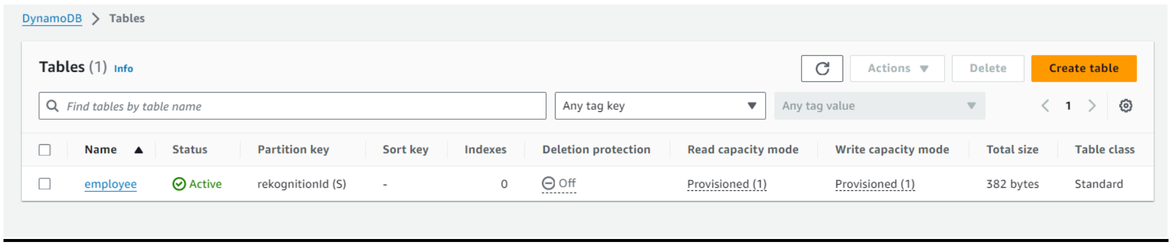
Path: /
Resource ID: 5mm8pkh5ci

Method type	Integration type	Authorization	API key

No methods

No methods defined.

4. **<u>DynamoDB:</u>** Amazon DynamoDB serves as the storage backbone for this face recognition application, offering a scalable and highly available NoSQL database solution. DynamoDB efficiently manages and stores crucial data related to visitor and employee information, ensuring quick and reliable access. Its seamless integration with other AWS services, such as AWS Lambda and Amazon S3, facilitates dynamic data retrieval and updates. Leveraging DynamoDB's flexibility, the application can handle varying workloads, adapting to changing storage requirements. With automatic scaling and low-latency performance, DynamoDB enhances the responsiveness of the face recognition system, providing a robust foundation for securely storing and retrieving data critical to the application's functionality.



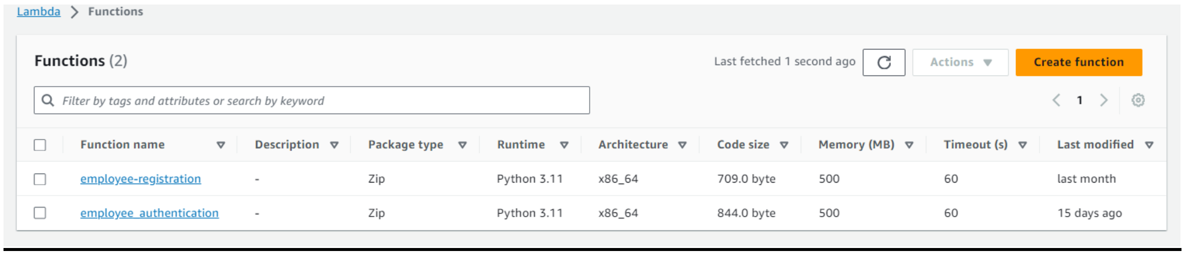
DynamoDB > Tables

Tables (1) Info

Find tables by table name | Any tag key | Any tag value

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size	Table class
employee	Active	rekognitionId (S)	-	0	Off	Provisioned (1)	Provisioned (1)	382 bytes	Standard

5. **<u>Lambda:</u>** AWS Lambda functions play a pivotal role in the face recognition application, serving both authentication and visitor face recognition. For authentication, Lambda efficiently handles user verification processes, ensuring secure access to the system. In the context of visitor face recognition, Lambda executes the recognition codes, leveraging the power of AWS Rekognition to identify and validate visitors. The serverless architecture of Lambda allows for seamless scalability, with code execution triggered in response to events, optimizing resource utilization. This integration enhances the application's efficiency by offloading computational tasks to Lambda, providing a dynamic and responsive solution for secure authentication and visitor face recognition.



Lambda > Functions

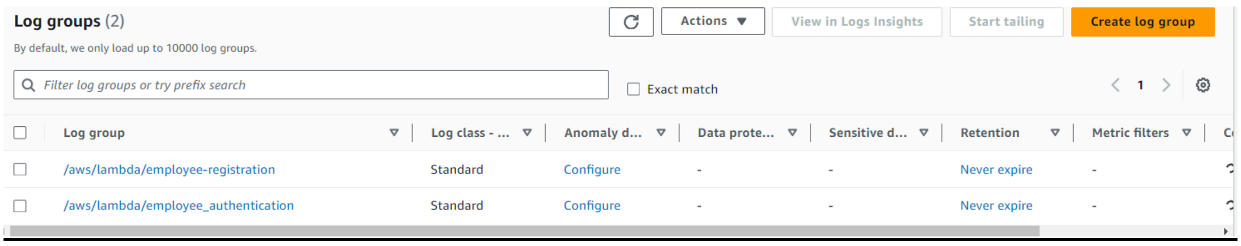
Functions (2)

Last fetched 1 second ago

Filter by tags and attributes or search by keyword

Function name	Description	Package type	Runtime	Architecture	Code size	Memory (MB)	Timeout (s)	Last modified
employee-registration	-	Zip	Python 3.11	x86_64	709.0 byte	500	60	last month
employee_authentication	-	Zip	Python 3.11	x86_64	844.0 byte	500	60	15 days ago

6. **<u>CloudWatch:</u>** AWS CloudWatch Logs serves as a comprehensive monitoring and logging solution for the face recognition application. It captures and stores log data generated by Lambda functions, API Gateway, and other AWS services, providing valuable insights into system behavior, performance, and user interactions. With CloudWatch Logs, administrators can analyze, filter, and search log information in real-time, aiding in troubleshooting and performance optimization. Its integration with AWS services allows for the creation of alarms and automated responses based on specific log events, ensuring proactive monitoring. CloudWatch Logs serves as a central hub for tracking and managing application activities, contributing to the overall health and security of the system.



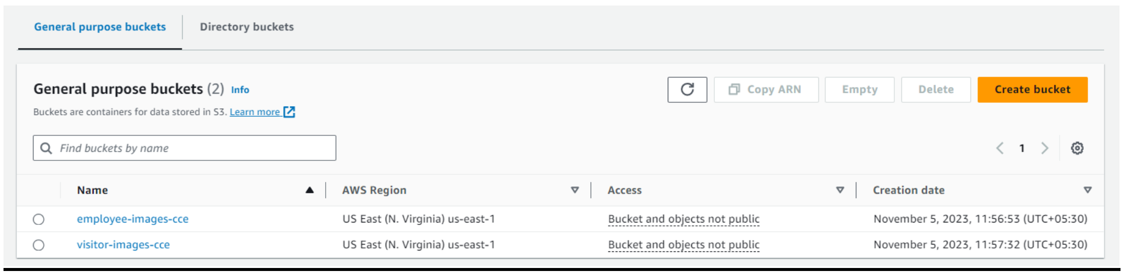
Log groups (2)

By default, we only load up to 10000 log groups.

Filter log groups or try prefix search | Exact match

Log group	Log class - ...	Anomaly d...	Data prote...	Sensitive d...	Retention	Metric filters
/aws/lambda/employee-registration	Standard	Configure	-	-	Never expire	-
/aws/lambda/employee_authentication	Standard	Configure	-	-	Never expire	-

7. **S3 Bucket:** The utilization of two distinct Amazon S3 buckets is a strategic choice in the face recognition application. One S3 bucket is dedicated to storing employee images, facilitating efficient retrieval and management of personnel data. Simultaneously, a separate S3 bucket is allocated for visitor images, ensuring a clear segregation of data for streamlined access control. This approach enhances scalability and allows for specific handling of employee and visitor information. Leveraging the reliability and scalability of S3, the dual-bucket setup optimizes storage and retrieval processes, contributing to the seamless functionality of the face recognition system while maintaining a structured and organized data storage architecture.



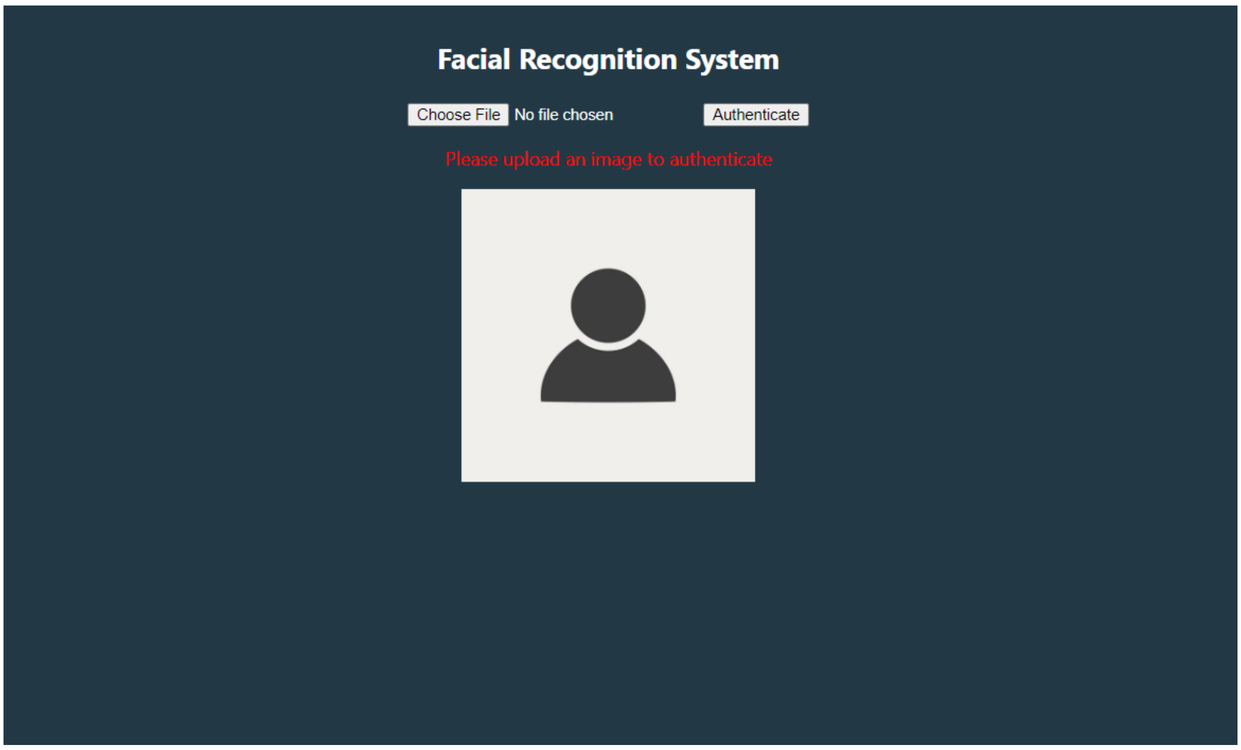
General purpose buckets (2) Info

Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
employee-images-cce	US East (N. Virginia) us-east-1	Bucket and objects not public	November 5, 2023, 11:56:53 (UTC+05:30)
visitor-images-cce	US East (N. Virginia) us-east-1	Bucket and objects not public	November 5, 2023, 11:57:32 (UTC+05:30)

UI DESIGN AND ITS SCREENSHOTS:

The webpage presents a Facial Recognition System interface built using React and AWS services. Users can upload visitor images for authentication. The website communicates with AWS Lambda functions and API Gateway to store and authenticate images in S3 buckets. The React app manages state, displaying authentication results dynamically. The user-friendly design incorporates a dark theme, aligning with a professional aesthetic. Authentication outcomes trigger color-coded messages, with success in green and failure in red. The webpage offers a seamless experience, blending functionality with visually appealing elements, making it an effective and intuitive tool for facial recognition authentication in an office setting.



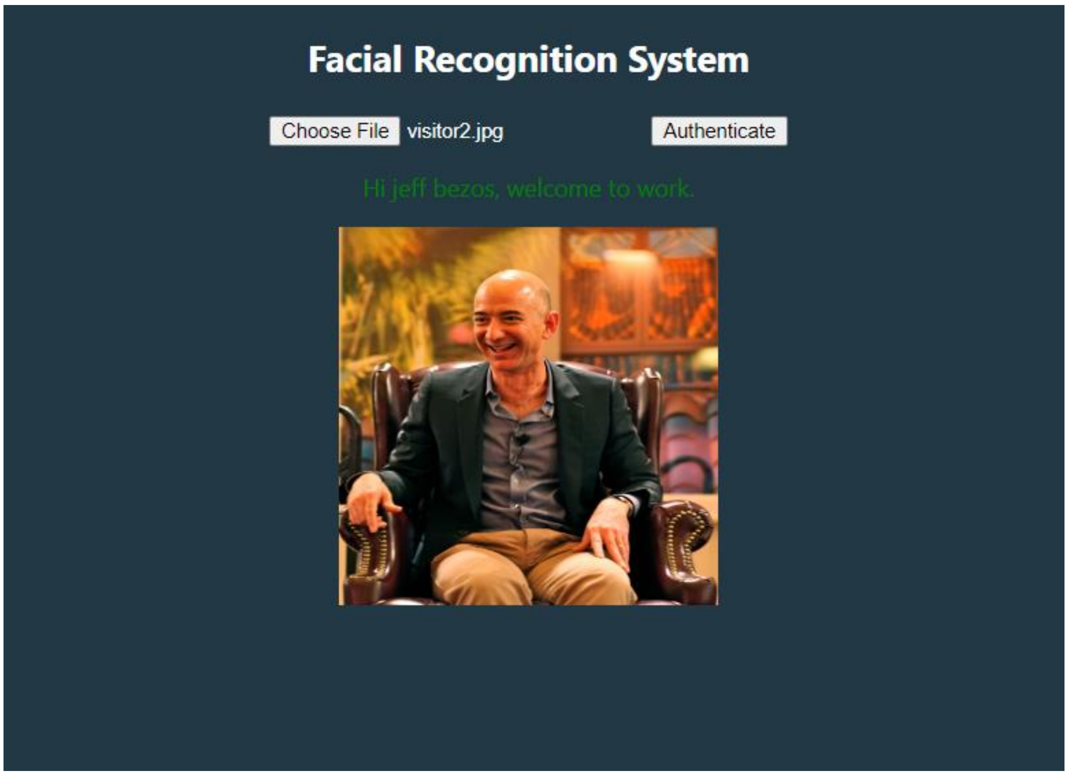
Facial Recognition System

No file chosen

Please upload an image to authenticate

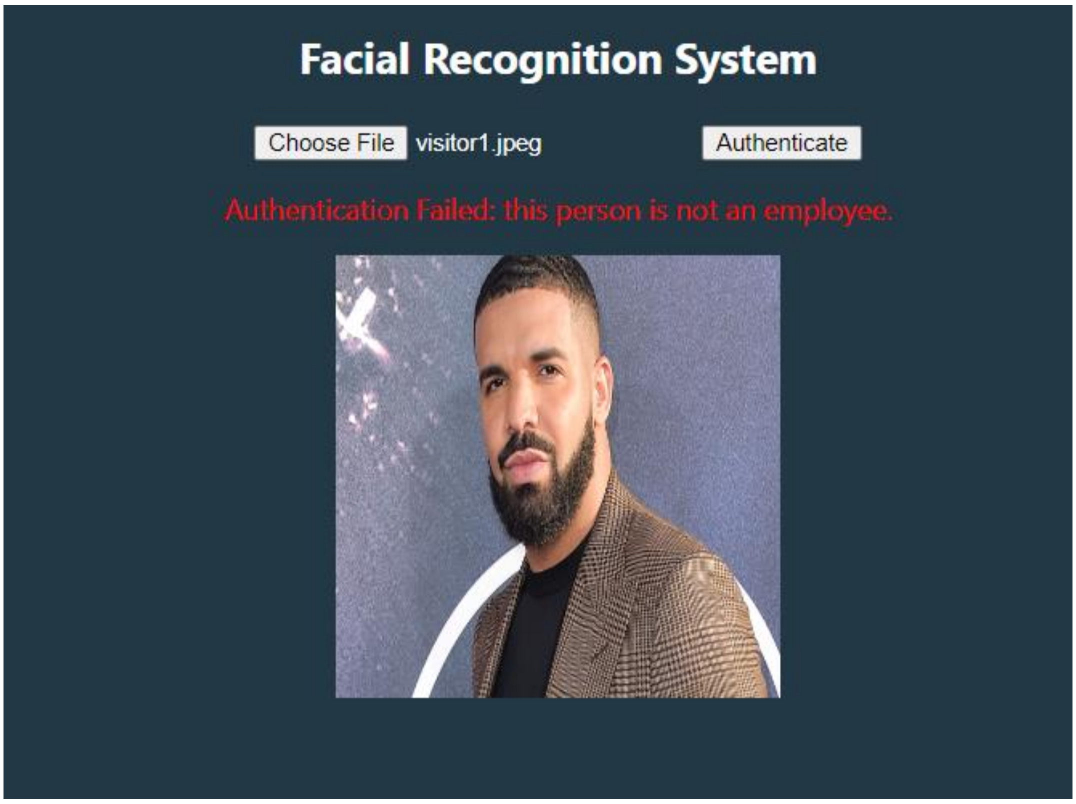
OUTPUT PHOTOS OF THE WEBPAGE:

1. Output if the visitor is an employee



Hi jeff bezos, welcome to work.

2. Output if the visitor is not an employee



Authentication Failed: this person is not an employee.

CODES:

1. Lambda code for employee-registration:

```
import boto3
s3 = boto3.client('s3')
rekognition = boto3.client('rekognition', region_name = 'us-east-1')
dynamodbTableName = 'employee'
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
employeeTable = dynamodb.Table(dynamodbTableName)

def lambda_handler(event, context):
   print(event)
   bucket = event['Records'][0]['s3']['bucket']['name']
   key = event['Records'][0]['s3']['object']['key']

   try:
      response = index_employee_image(bucket, key)
      print(response)
      if response['ResponseMetadata']['HTTPStatusCode'] == 200:
         faceId = response['FaceRecords'][0]['Face']['FaceId']
         name = key.split('.')[0].split('_')

         firstName = name[0]
         lastName = name[1]
         register_employee(faceId, firstName, lastName)
      return response
   except Exception as e:
      print(e)
      print('Error processing employee image {} from bucket{}.'.format(key, bucket))
      raise e

def index_employee_image(bucket, key):
   response = rekognition.index_faces(
      Image={
         'S3Object':
         {
            'Bucket' : bucket,
            'Name' : key
         }
      },
      CollectionId="employees"
   )
   return response

def register_employee(faceId, firstName, lastName):
   employeeTable.put_item(
      Item={
         'rekognitionId' : faceId,
         'firstName' : firstName,
         'lastName' : lastName
      }
   )
```

2. Lambda code for employee-authentication:

```
import boto3
import json

s3 = boto3.client('s3')
rekognition = boto3.client('rekognition', region_name = 'us-east-1')
dynamodbTableName = 'employee'
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
employeeTable = dynamodb.Table(dynamodbTableName)
bucketName = 'visitor-images-cce'

def lambda_handler(event, context):
    print(event)
    objectKey = event.get('queryStringParameters', {}).get('objectKey')
    print("Object Key:", objectKey)
    image_bytes = s3.get_object(Bucket=bucketName, Key=objectKey)['Body'].read()
    print(image_bytes)
    response = rekognition.search_faces_by_image(
        CollectionId='employees',
        Image={'Bytes':image_bytes}    )

    for match in response['FaceMatches']:
        print(match['Face'] ['FaceId'], match['Face']['Confidence'])

        face = employeeTable.get_item(
            Key={
                'rekognitionId' : match['Face']['FaceId']
            }
        )
        if 'Item' in face:
            print('Person Found: ', face['Item'])
            return buildResponse(200, {
                'Message' : 'Success',
                'firstName' : face['Item']['firstName'],
                'lastName' : face['Item']['lastName']
            })
    print('Person could not be recognized.')
    return buildResponse(403, {'Message': 'Person Not Found'})

def buildResponse(statusCode, body=None):
    response = {
        'statusCode': statusCode,
        'headers': {
            'Access-Control-Allow-Origin' : '*',
            'Content-Type': 'application/json'

        }
    }
    if body is not None:
        response['body'] = json.dumps(body)
        return response
```

3. **Code for App.css:**

```
body {
  background-color: rgb(34, 57, 69);
  color: white; /* Set text color to white */
  margin: 0;
  padding: 0;
  text-decoration: rgb(205, 212, 212);
}

.App {
  margin-top: 2rem;
  text-align: center;
}

.success {
  padding: 1rem;
  color: green;
}

.failure {
  padding: 1rem;
  color: red;
}
```

4. **Code for App.js:**

```
import { useState } from 'react';
import './App.css';
const uuid = require('uuid');

function App() {
 const [image, setImage] = useState('');
 const [uploadResultMessage, setUploadResultMessage] = useState('Please upload an
image to authenticate');
 const[visitorName, setVisitorName] = useState('placeholder.jpg');
 console.log(visitorName);
 const [isAuth, setAuth] = useState(false)

 function sendImage(e){
  e.preventDefault();
  setVisitorName(image.name);
  const visitorImageName = uuid.v4();
  fetch(`https://5u3s1uz1ml.execute-api.us-east-1.amazonaws.com/dev1/visitor-
images-cce/${visitorImageName}.jpeg`, {

   method: 'PUT',
   headers: {
    'Content-Type': 'image/jpeg'
   },
   body: image
  }).then(async () => {
   const response = await authenticate(visitorImageName)
   if (response.Message === 'Success') {
    setAuth(true);
    setUploadResultMessage(`Hi ${response['firstName']} ${response['lastName']},
welcome to work.`)
     } else {
       setAuth(false);
       setUploadResultMessage('Authentication Failed: this person is not an employee.')
     }
   }).catch(error => {
     setAuth(false);
     console.log(error);
     setUploadResultMessage('There is an error during the authentication process. Please
try again.')
     console.error(error);
   })
 }

 async function authenticate(visitorImageName) {
   const        requestUrl        =        'https://5u3s1uz1ml.execute-api.us-east-
1.amazonaws.com/dev1/employee?' + new URLSearchParams({
     objectKey: `${visitorImageName}.jpeg`
   });
   console.log(`${visitorImageName}.jpeg`);
   return await fetch(requestUrl, {
     method: 'GET',
     headers: {
       'Accept': 'application/json',
       'Content-Type': 'application/json'
         }
   }).then(response => response.json())
   .then((data) => {
     console.log(data);
     return data;
   }).catch(error => console.error(error));
 }

 return (
   <div className="App">

     <h2>Facial Recognition System</h2>

     <form onSubmit={sendImage}>
       <input type='file' name='image' onChange={e => setImage(e.target.files[0])}/>
       <button type='submit'>Authenticate</button>
     </form>
     <div className={isAuth ? 'success' : 'failure' }>{uploadResultMessage}</div>
     <img    src={require(`./visitors/${visitorName}`)}    alt="Visitor"    height={250}
width={250}/>
   </div>
 );
}

export default App;
```

5. Code for index.css:

```css
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```