# Cloud Computing Final Project Report

Automated Facial verification with Zoom integration

Bomin Zhang, Dakeun Park, Edmund Park, YiWei, Zeren Li
(In alphabetical order)

## Index

1. Work Repository Links
2. Members and Contributions
3. Abstract
4. Problem Statement
5. Significance
6. Implementation Tools
7. General Workflow
8. Evaluation Results
9. Conclusion
10. Related Works

## Work Repository Links

Original Intended Implementation of Verification App(Zoom App): https://github.com/RealmX1/MSML650-coud-computing-final-proj.git

Web App URL:

https://main.dqruf5i3ks8x6.amplifyapp.com/

Web App github repository:

https://github.com/zli868/amplify-vite-react-template/tree/main

## Members and Contributions

- **Yi Wei (ywei1234@umd.edu)**

  **Role**: Lead Backend Developer

  **Planned Contribution**: Yi will focus on integrating AWS Rekognition with the backend system. Utilizing his experience in C++, Python, and C#, he will develop the core algorithms for sending image frames to AWS Rekognition and processing the results. His background in NLP and RL applications will aid in optimizing the recognition process.

- **Bomin Zhang (bominz2@umd.edu)**

  **Role**: Frontend Development

  **Planned Contribution**: Bomin will handle the development of the user interface and dashboard for displaying recognition results in real-time. A crude interface has already been developed – one for user registration and one for meeting hosts to see verification status of each attendee.

- **Dakeun (Dan) Park (dpark37@umd.edu)**

  **Role**: Cloud Infrastructure Engineer

  **Planned Contribution**: Dan will set up and manage the AWS infrastructure, including S3 storage, Lambda functions, and API Gateway. His experience with AWS services and low-code platforms will ensure a scalable and efficient deployment. He will also implement notifications using AWS SNS and integrate them with the system.

- **Edmund Park (epark12@umd.edu)**

  **Role**: Quality Assurance and Testing Specialist

  **Planned Contribution**: Edmund will be responsible for testing the system at each development stage. He will create test cases to ensure the face recognition system functions correctly and securely. His attention to detail will help in identifying and fixing bugs promptly.

- **Zeren Li (zli06211@umd.edu)**

  **Role**: Workflow designer and Web app developer

**Contribution**: Zeren was involved and created a pipeline for our application which served as the basis for our discussion about the framework. He will also focus on the creation of the web app, which leverages the user login system by Amazon Cognito. Main functionality of the web app includes services that allow our client users to upload their images, and take a selfie on the web app, and finally register our user's face information in the Amazon Rekognition database, which is a collection. The app will be hosted on Amazon amplify, so the service will be available in the public network.

## Abstract

This project implements an automated Facial Verification Service that Integrates with Zoom through video stream, using AWS Amplify for front end hosting and integrated user registration, AWS Rekognition for face verification, and AWS lambda for backend. It allows the host to verify that attendees of the meeting are valid users that have their face data stored in the system.

## Problem Statement

In virtual meetings, verifying participant identities is a challenge, leading to potential security risks such as unauthorized access or taking accountability of participants. This means online meetings are vulnerable to access from unauthorized or even malicious actors, potentially joining the meeting from an invited registered user account. There is a lack of real-time, automated systems to authenticate participants in video teleconferencing (VTC) platforms like Zoom. Our project aims to solve this by integrating face recognition technology to ensure accountability of authorized participants.

## Significance

Provided automated attendants face verification framework, which can be later built upon to prevent unverified persons from joining the meeting, or utilize a more feature rich facial recognition system to track attendants’ attentiveness to the meeting.

## Implementation tools

Please include any softcopy artifacts, i.e., source code, database with your submission, which may be in the form of a URL to an external repository. One submission per group is sufficient.

We used AWS Amplify for hosting the Cognito User Pool, Registration UI, and host verification UI. We also used AWS Rekognition for registering user faces, and for matching user faces to registered user ID during meetings when Host activated the functionality. Zoom Stream is used to provide user facial information to AWS Rekognition, and AWS Lambda & API GateWay were used to connect all of these functionality. Finally, the SNS email service is used to notify the host about the verification result.

We originally planned to use ZoomApp API, and we finished with UI design and authorization routing building. but we met a dead end when debugging the http request making from a zoom authorized app using zoom's template, and despite lengthy effort to address this issue, we eventually went with the current tool setup.

### 1. Lambda Code

I. FaceDataCapture:

```python
import boto3
import json
import base64
from botocore.exceptions import ClientError

rekognition_client = boto3.client('rekognition')

def lambda_handler(event, context):
    print(f"Incoming event: {json.dumps(event)}")

    # Handle CORS preflight requests
    if event['httpMethod'] == 'OPTIONS':
        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Methods': 'POST, OPTIONS',
                'Access-Control-Allow-Headers': 'Content-Type, x-user-email',
            },
            'body': json.dumps("CORS preflight response"),
        }

    try:
        # Extract email from headers
        email = event['headers'].get('x-user-email')
        if not email:
            return {
                'statusCode': 400,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps("Missing 'x-user-email' header.")
            }

        # Extract binary image data from the body
        image_bytes = event['body']
        if event['isBase64Encoded']:
            image_bytes = base64.b64decode(image_bytes)

        collection_id = 'face-collection'  # Your Rekognition collection ID

        # Step 1: Create a user in Rekognition using the email as user ID
        user_created = False
        try:
            create_user_response = rekognition_client.create_user(
                CollectionId=collection_id,
                UserId=email  # Use email as the User ID
            )
            user_created = True
            print(f"User created successfully: {create_user_response}")
        except ClientError as e:
            error_message = e.response['Error']['Message']
            print(f"AWS Rekognition error during user creation:
{error_message}")
            return {
                'statusCode': 500,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps(f"User already Exist: {error_message}")
            }

        # Step 2: Index the face in the Rekognition collection
        try:
            index_faces_response = rekognition_client.index_faces(
                CollectionId=collection_id,
                Image={'Bytes': image_bytes},
                DetectionAttributes=['ALL'],
                ExternalImageId=email  # Use the email as an external
identifier
            )
            print(f"Index faces response: {index_faces_response}")
        except ClientError as e:
            error_message = e.response['Error']['Message']
            print(f"AWS Rekognition error during indexing: {error_message}")

            # Rollback: Delete the user created in Step 1
            if user_created:
                print(f"Rolling back: Deleting user '{email}' due to failure in
indexing.")
                try:
                    rekognition_client.delete_user(CollectionId=collection_id,
UserId=email)
                    print(f"User '{email}' deleted successfully.")
                except ClientError as rollback_error:
                    print(f"Failed to rollback user deletion:
{rollback_error.response['Error']['Message']}")

            return {
                'statusCode': 500,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps(f"Failed to index face: {error_message}")
            }

        # Check if any face was indexed
        if not index_faces_response.get('FaceRecords'):
            # Rollback: Delete the user created in Step 1
            if user_created:
                print(f"Rolling back: Deleting user '{email}' due to no face
detected.")
                try:
                    rekognition_client.delete_user(CollectionId=collection_id,
UserId=email)
                    print(f"User '{email}' deleted successfully.")
                except ClientError as rollback_error:
                    print(f"Failed to rollback user deletion:
{rollback_error.response['Error']['Message']}")
```

```
            return {
                'statusCode': 400,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps("No face could be indexed in the provided
image.")
            }

        # Extract the indexed face ID
        face_id = index_faces_response['FaceRecords'][0]['Face']['FaceId']

        # Step 3: Associate the indexed face with the created user
        try:
            associate_faces_response = rekognition_client.associate_faces(
                CollectionId=collection_id,
                UserId=email,
                FaceIds=[face_id]
            )
            print(f"Face associated with user successfully:
{associate_faces_response}")
        except ClientError as e:
            error_message = e.response['Error']['Message']
            print(f"AWS Rekognition error during face association:
{error_message}")
            return {
                'statusCode': 500,
                'headers': {'Access-Control-Allow-Origin': '*'},
                'body': json.dumps(f"Failed to associate face:
{error_message}")
            }

        # Successful response
        return {
            'statusCode': 200,
            'headers': {'Access-Control-Allow-Origin': '*'},
            'body': json.dumps(f"Face associated with user '{email}'
successfully.")
        }

    except Exception as e:
        print(f"Unexpected Error: {str(e)}")
        return {
            'statusCode': 500,
            'headers': {'Access-Control-Allow-Origin': '*'},
            'body': json.dumps(f"An unexpected error occurred: {str(e)}")
        }
```

## II. RekoMatchUsersFromImage:

```
import boto3
import json
import logging
from PIL import Image
from io import BytesIO

# Initialize AWS clients
rekognition_client = boto3.client('rekognition')
s3_client = boto3.client('s3')

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Define CORS headers
    cors_headers = {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'POST, GET, OPTIONS',
        'Access-Control-Allow-Headers': 'Content-Type'
    }

    try:
        # Log the incoming event
        logger.info("Received event: %s", json.dumps(event, indent=2))

        # Extract bucket and object key from the S3 trigger event
        bucket = event['Records'][0]['s3']['bucket']['name']
        key = event['Records'][0]['s3']['object']['key']

        logger.info("Processing image from bucket: %s, key: %s", bucket, key)

        # Download the image from S3
        s3_response = s3_client.get_object(Bucket=bucket, Key=key)
        image_data = s3_response['Body'].read()
        logger.info("Successfully downloaded image from S3")

        # Call Rekognition to detect faces
        detect_faces_response = rekognition_client.detect_faces(
            Image={'Bytes': image_data},
            Attributes=['DEFAULT']
        )
        logger.info("DetectFaces response: %s",
json.dumps(detect_faces_response, indent=2))

        # Extract bounding boxes from the response
        face_details = detect_faces_response.get('FaceDetails', [])
        if not face_details:
            logger.warning("No faces detected in the image")
            return {
                'statusCode': 404,
                'headers': cors_headers,
                'body': json.dumps("No faces detected in the image.")
            }

        # Open the original image for processing
        image = Image.open(BytesIO(image_data))
        width, height = image.size
        logger.info("Image size: width=%d, height=%d", width, height)

        results = []
        for idx, face in enumerate(face_details):
            # Get bounding box and convert relative coordinates to absolute
coordinates
            bounding_box = face['BoundingBox']
            left = int(bounding_box['Left'] * width)
            top = int(bounding_box['Top'] * height)
            box_width = int(bounding_box['Width'] * width)
            box_height = int(bounding_box['Height'] * height)

            logger.info(
                "Face %d bounding box: left=%d, top=%d, width=%d, height=%d",
                idx + 1, left, top, box_width, box_height
            )

            # Crop the face from the image
            cropped_face = image.crop((left, top, left + box_width, top +
box_height))
            logger.info("Cropped face %d successfully", idx + 1)

            # Save the cropped face to memory
            cropped_face_bytes = BytesIO()
            cropped_face.save(cropped_face_bytes, format='JPEG')
            cropped_face_bytes = cropped_face_bytes.getvalue()

            # Call Rekognition to search for matching faces
            try:
                search_response = rekognition_client.search_users_by_image(
                    CollectionId='face-collection',  # Replace with your
Rekognition collection ID
                    Image={'Bytes': cropped_face_bytes},
                    MaxUsers=1,
                    UserMatchThreshold=95
                )
                logger.info("SearchUsersByImage response for face %d: %s", idx
+ 1, json.dumps(search_response, indent=2))

                user_matches = search_response.get('UserMatches', [])
                if user_matches:
                    matched_user = user_matches[0]
                    logger.info("Matched user for face %d: %s", idx + 1,
matched_user['User']['UserId'])
                    results.append({
                        'user_id': matched_user['User']['UserId'],
                        'similarity': matched_user['Similarity'],
                        'face_index': idx
                    })
                else:
                    logger.info("No matching user found for face %d", idx + 1)
                    results.append({
                        'message': f"No matching user found for face {idx +
1}.",
                        'face_index': idx
                    })
            except rekognition_client.exceptions.ClientError as e:
                error_message = e.response['Error']['Message']
                logger.error("Error during SearchUsersByImage for face %d: %s",
idx + 1, error_message)
                results.append({
                    'error': f"Error during SearchUsersByImage for face {idx +
1}: {error_message}",
                    'face_index': idx
                })

        # Log and return all results
        logger.info("Processing results: %s", json.dumps(results, indent=2))
        return {
            'statusCode': 200,
            'headers': cors_headers,
            'body': json.dumps(results)
        }
```

```
    except Exception as e:
        error_message = str(e)
        logger.error("Error processing image: %s", error_message)
        return {
            'statusCode': 500,
            'headers': cors_headers,
            'body': json.dumps({"error": error_message})
        }
```

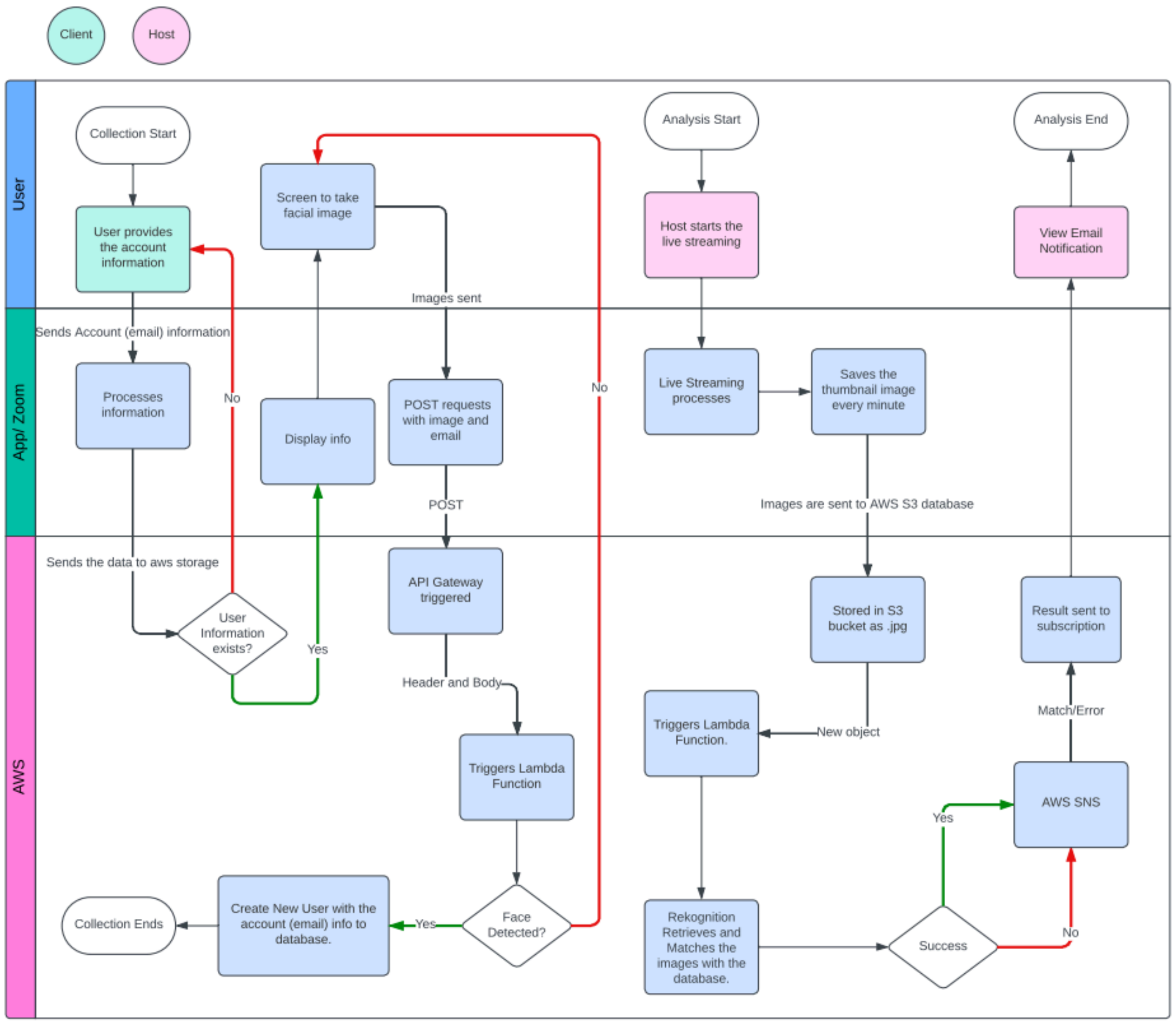
## 2. Web-App Hosted on Amplify

### I. main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { Authenticator } from '@aws-amplify/ui-react';
import App from "./App.tsx";
import "./index.css";
import { Amplify } from "aws-amplify";
import outputs from "../amplify_outputs.json";
import '@aws-amplify/ui-react/styles.css';

Amplify.configure(outputs);

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <Authenticator>
      <App />
    </Authenticator>
  </React.StrictMode>
);

App.tsx
import React, { useState, useRef } from "react";
import { useAuthenticator } from '@aws-amplify/ui-react';
import "./styles.css";

function App() {
  const { user } = useAuthenticator((context) => [context.user]);
  const [uploadedImage, setUploadedImage] = useState<File | null>(null); //
Explicit type
  const [capturedImage, setCapturedImage] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);
  const [responseMessage, setResponseMessage] = useState("");
  const videoRef = useRef<HTMLVideoElement | null>(null);
  const canvasRef = useRef<HTMLCanvasElement | null>(null);
  const { signOut } = useAuthenticator();

  // Handles the file input change
  const handleImageChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    if (event.target.files && event.target.files[0]) {
      setUploadedImage(event.target.files[0]);
    }
  };

  const startCamera = async () => {
    if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
      try {
        const stream = await navigator.mediaDevices.getUserMedia({ video: true
});
        if (videoRef.current) {
          videoRef.current.srcObject = stream;
        }
      } catch (error) {
        console.error("Error accessing the camera", error);
      }
    }
  };

  const captureSelfie = () => {
    if (videoRef.current && canvasRef.current) {
      const context = canvasRef.current.getContext("2d");
      if (context) {
        context.drawImage(videoRef.current, 0, 0, 640, 480);
        const imageDataUrl = canvasRef.current.toDataURL("image/png");
        setCapturedImage(imageDataUrl);
      }
    }
  };

  const handleUpload = async () => {
    if (!uploadedImage && !capturedImage) {
      alert("Please select an image or capture a selfie.");
      return;
    }
    if (!user) {
      alert("Please make sure you're logged in.");
      return;
    }

    setLoading(true);
    setResponseMessage("");

    try {
      const userEmail = user.signInDetails?.loginId || '';
      const modifiedEmail = userEmail.replace('@', ':');
      const imageBlob = capturedImage ? await (await
fetch(capturedImage)).blob() : uploadedImage;

      // Make the API call to Lambda
      const response = await fetch(

"https://v8c6qwk16b.execute-api.us-east-1.amazonaws.com/default/FaceDataCapture
",
        {
          method: "POST",
          headers: {
            'x-user-email': modifiedEmail,
          },
          body: imageBlob,
        }
      );

      const responseData = await response.json();

      if (response.ok) {
        setResponseMessage(
          "Registration Successful: " + JSON.stringify(responseData)
        );
      } else {
        setResponseMessage("Error: " + JSON.stringify(responseData));
      }
    } catch (error) {
      if (error instanceof Error) {
        console.error("Upload error", error);
        setResponseMessage("Upload error: " + error.message);
      } else {
        console.error("Unexpected error", error);
        setResponseMessage("An unexpected error occurred.");
      }
    } finally {
      setLoading(false);
    }
```

```
  };

  return (
    <div className="App">
      <h2>Register Your Face</h2>
      <p>Email: {user.signInDetails?.loginId}</p>
      <div>
        <input type="file" accept="image/*" onChange={handleImageChange} />
      </div>
      <div>
        <button onClick={startCamera}>Start Camera</button>
        <video ref={videoRef} width="640" height="480" autoPlay></video>
        <button onClick={captureSelfie}>Capture Selfie</button>
      </div>
      {capturedImage && (
        <div>
          <h3>Captured Selfie:</h3>
          <img src={capturedImage} alt="Captured Selfie" />
        </div>
      )}
      <button onClick={handleUpload} disabled={loading}>
        {loading ? "Uploading..." : "Upload Image"}
      </button>
      {responseMessage && <p>{responseMessage}</p>}
      <canvas ref={canvasRef} width="640" height="480" style={{ display: 'none'
}}></canvas>

      <button className="sign-out" onClick={signOut}>Sign out</button>

    </div>

  );
}

export default App;
```

# General Workflow



Client
Host
User
App/ Zoom
AWS
Collection Start
User provides the account information
Sends Account (email) information
Processes information
Sends the data to aws storage
User Information exists?
No
Yes
Display info
Screen to take facial image
Images sent
POST requests with image and email
POST
API Gateway triggered
Header and Body
Triggers Lambda Function
Face Detected?
No
Yes
Create New User with the account (email) info to database.
Collection Ends
Analysis Start
Host starts the live streaming
Live Streaming processes
Saves the thumbnail image every minute
Images are sent to AWS S3 database
Stored in S3 bucket as .jpg
New object
Triggers Lambda Function.
Rekognition Retrieves and Matches the images with the database.
Success
Yes
No
AWS SNS
Match/Error
Result sent to subscription
View Email Notification
Analysis End

## Evaluation results

Our application was initially designed to use a web app to register our users' face information and then use a self-designed zoom app to make the verification process. However, It seems like the zoom app prevents us from accessing the external APIs other than the APIs provided by the ZoomSdk. As a result, we cannot call the API that we made for face information verification and we regard this as an internal flaw in the zoom app. As an alternative approach, we used an existing zoom app called live streaming, which captures our face images in the camera and sends them out to the API which does the verification purposes. However, all the main services including the registration and verification are working as intended.

Still, the current system implements the core functionality to capture face data from a web application using AWS services like amplify, gateway, lambda. Moreover, using zoom's live stream service to capture thumb images of the meetings by AWS IVS to store in the S3 bucket and process the images for matching using lambda, and then sending the result with AWS SNS service.

### Web App Evaluation

Below are link to the web app hosted on Amplify along with the link to the github repository

Web App URL:

https://main.dqruf5i3ks8x6.amplifyapp.com/

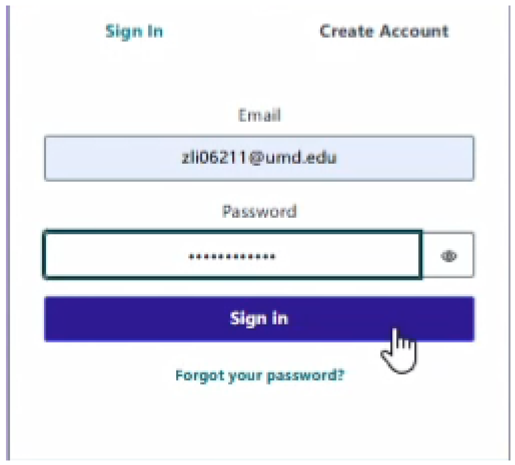
Web App github repository:

https://github.com/zli868/amplify-vite-react-template/tree/main

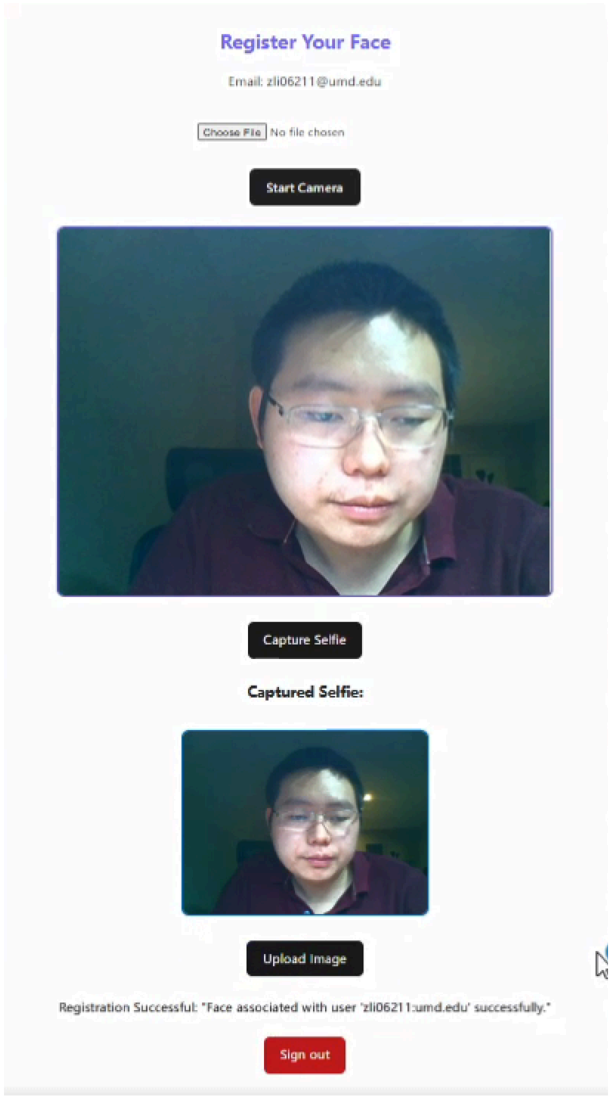
Web App live example:

🎬 Web-app-usage-demo.mp4

Once we go to the app, we will be prompted to login. Create a user if you have not already registered. The email will be associated with your identity.



In the application, you will be asked to either provide a self image stored on your local machine or to take a selfie and then upload. Upon clicking the upload image, a circle on the button will appear, which indicates that the image is uploading. Once done, it will return a message like “Successful” if registered or other error messages indicating the reasons why it was not successful.

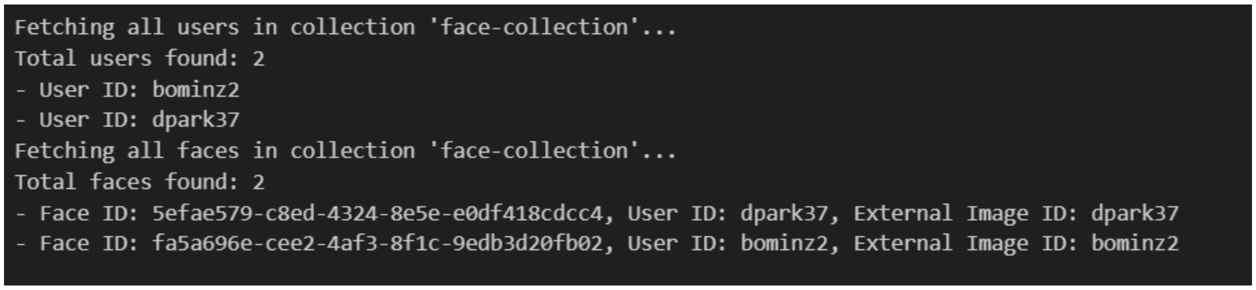


Registration Successful: "Face associated with user 'zli06211:umd.edu' successfully."

## Verification Process Evaluation

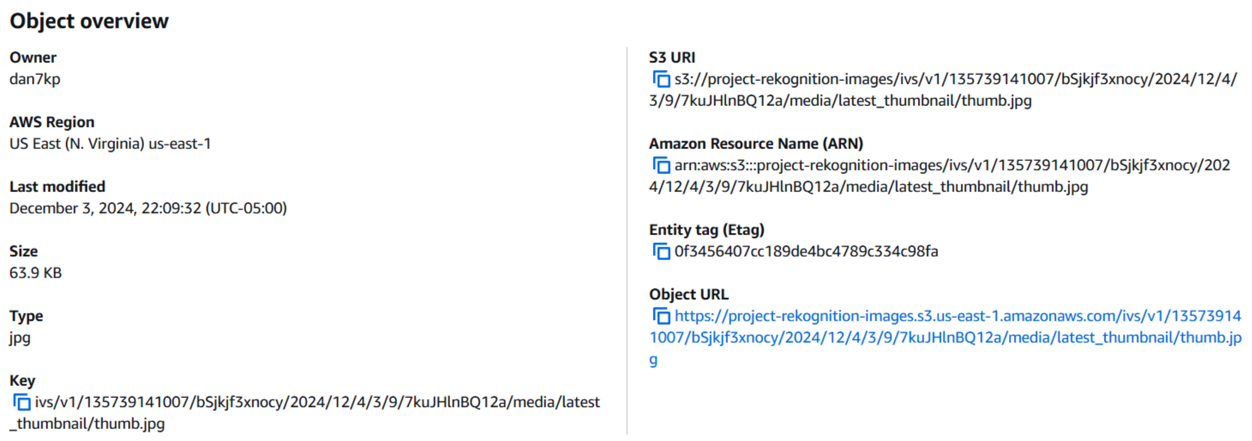
The evaluation of the AWS Rekognition successfully matches the stored user's face data with newly provided faces from the zoom meetings using the listed AWS services above.

Usage Example 1 - Two Users “dpark37” and “bominz2” registered as users in the Rekognition Collection.



```
Fetching all users in collection 'face-collection'...
Total users found: 2
- User ID: bominz2
- User ID: dpark37
Fetching all faces in collection 'face-collection'...
Total faces found: 2
- Face ID: 5efae579-c8ed-4324-8e5e-e0df418cdcc4, User ID: dpark37, External Image ID: dpark37
- Face ID: fa5a696e-cee2-4af3-8f1c-9edb3d20fb02, User ID: bominz2, External Image ID: bominz2
```

Zoom stores the thumbnail images to S3 using Live Stream with AWS IVS.



**Object overview**

**Owner**
dan7kp

**AWS Region**
US East (N. Virginia) us-east-1

**Last modified**
December 3, 2024, 22:09:32 (UTC-05:00)

**Size**
63.9 KB

**Type**
jpg

**Key**
ivs/v1/135739141007/bSjkjf3xnocy/2024/12/4/3/9/7kuJHlnBQ12a/media/latest_thumbnail/thumb.jpg

**S3 URI**
s3://project-rekognition-images/ivs/v1/135739141007/bSjkjf3xnocy/2024/12/4/3/9/7kuJHlnBQ12a/media/latest_thumbnail/thumb.jpg

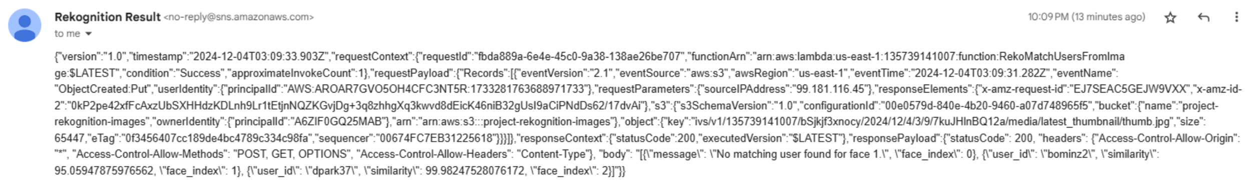
**Amazon Resource Name (ARN)**
arn:aws:s3:::project-rekognition-images/ivs/v1/135739141007/bSjkjf3xnocy/2024/12/4/3/9/7kuJHlnBQ12a/media/latest_thumbnail/thumb.jpg

**Entity tag (Etag)**
0f3456407cc189de4bc4789c334c98fa

**Object URL**
https://project-rekognition-images.s3.us-east-1.amazonaws.com/ivs/v1/135739141007/bSjkjf3xnocy/2024/12/4/3/9/7kuJHlnBQ12a/media/latest_thumbnail/thumb.jpg

Tested the unregistered user’s face for verification.

The message which contains all participants will be forwarded to the hosts' email. As you can see from the message, only the face with associated users are matched sent from AWS SNS.


Rekognition Result <no-reply@sns.amazonaws.com>
to me
10:09 PM (13 minutes ago)

```
{"version":"1.0","timestamp":"2024-12-04T03:09:33.903Z","requestContext":{"requestId":"fbda889a-6e4e-45c0-9a38-138ae26be707","functionArn":"arn:aws:lambda:us-east-1:135739141007:function:RekoMatchUsersFromIma
ge:$LATEST","condition":"Success","approximateInvokeCount":1},"requestPayload":{"Records":[{"eventVersion":"2.1","eventSource":"aws:s3","awsRegion":"us-east-1","eventTime":"2024-12-04T03:09:31.282Z","eventName":
"ObjectCreated:Put","userIdentity":{"principalId":"AWS:AROAR7GVO5OH4CFC3NT5R:1733281763688971733"},"requestParameters":{"sourceIPAddress":"99.181.116.45"},"responseElements":{"x-amz-request-id":"EJ7SEAC5GEJW9VXX","x-amz-id-
2":"0kP2pe42xfFcAxzUbSXHHdzKDLnh9Lr1tEtjnNQZKGvjDg+3q8zhhgXq3kwvd8dEicK46niB32gUsI9aCiPNdDs62/17dvAi"},"s3":{"s3SchemaVersion":"1.0","configurationId":"00e0579d-840e-4b20-9460-a07d748965f5","bucket":{"name":"project-
rekognition-images","ownerIdentity":{"principalId":"A6ZIF0GQ25MAB"},"arn":"arn:aws:s3:::project-rekognition-images"},"object":{"key":"ivs/v1/135739141007/bSjkjf3xnocy/2024/12/4/3/9/7kuJHlnBQ12a/media/latest_thumbnail/thumb.jpg","size":
65447,"eTag":"0f3456407cc189de4bc4789c334c98fa","sequencer":"00674FC7EB31225618"}}}]},"responseContext":{"statusCode":200,"executedVersion":"$LATEST"},"responsePayload":{"statusCode": 200, "headers": {"Access-Control-Allow-Origin":
"*", "Access-Control-Allow-Methods": "POST, GET, OPTIONS", "Access-Control-Allow-Headers": "Content-Type"}, "body": "[{\"message\": \"No matching user found for face 1.\", \"face_index\": 0}, {\"user_id\": \"bominz2\", \"similarity\":
95.05947875976562, \"face_index\": 1}, {\"user_id\": \"dpark37\", \"similarity\": 99.98247528076172, \"face_index\": 2}]"}}
```

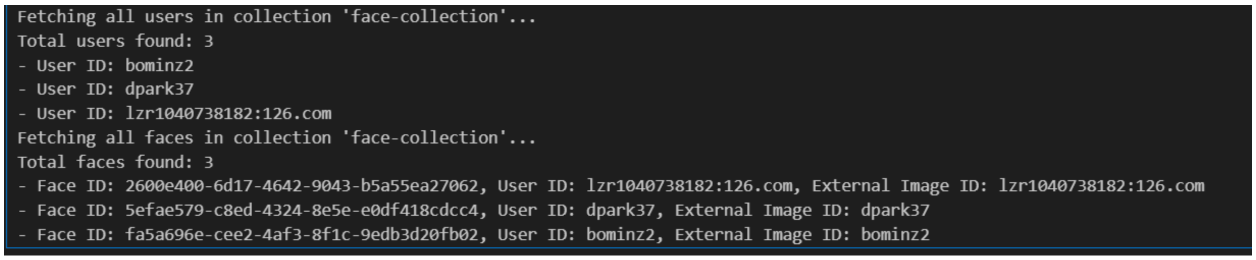
```
Allow-Methods": "POST, GET, OPTIONS", "Access-Control-Allow-Headers": "Content-Type"}, "body": "
[{\"message\": \"No matching user found for face 1.\", \"face_index\": 0}, {\"user_id\": \"bominz2\",
\"similarity\": 95.05947875976562, \"face_index\": 1}, {\"user_id\": \"dpark37\", \"similarity\":
99.98247528076172, \"face_index\": 2}]"}}
```

Face index 0 shows "No matching user found for face 0." and for face index 1 and 2 shows the corresponding matching faces with a high similarity (similarity above 95 percent means match is found!).

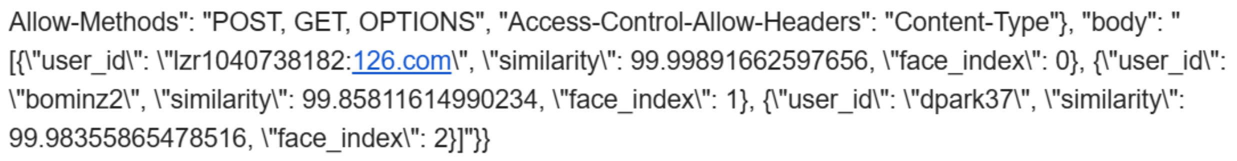
## Usage Example 2 - Live demo

650_example.mp4

3 faces registered as Users in the collection.



```
Fetching all users in collection 'face-collection'...
Total users found: 3
- User ID: bominz2
- User ID: dpark37
- User ID: lzr1040738182:126.com
Fetching all faces in collection 'face-collection'...
Total faces found: 3
- Face ID: 2600e400-6d17-4642-9043-b5a55ea27062, User ID: lzr1040738182:126.com, External Image ID: lzr1040738182:126.com
- Face ID: 5efae579-c8ed-4324-8e5e-e0df418cdcc4, User ID: dpark37, External Image ID: dpark37
- Face ID: fa5a696e-cee2-4af3-8f1c-9edb3d20fb02, User ID: bominz2, External Image ID: bominz2
```



```
Allow-Methods": "POST, GET, OPTIONS", "Access-Control-Allow-Headers": "Content-Type"}, "body": "
[{\"user_id\": \"lzr1040738182:126.com\", \"similarity\": 99.99891662597656, \"face_index\": 0}, {\"user_id\":
\"bominz2\", \"similarity\": 99.85811614990234, \"face_index\": 1}, {\"user_id\": \"dpark37\", \"similarity\":
99.98355865478516, \"face_index\": 2}]"}}
```

The result email from AWS SNS shows that all three members in the meeting are matching members(similarity > 99). (if there exists an unregistered member, the result would be “did not find a match for {user_id}”.

## Conclusion

This project report presents the development and implementation of an **Automated Facial Verification System with Zoom Integration**, leveraging AWS cloud services. The primary objective of the project is to address the security challenge of verifying participant identities in virtual meetings by integrating facial recognition technology into Zoom. Using AWS Rekognition, Amplify, Lambda, and SNS, the system enables meeting hosts to validate participants in real-time against a pre-registered facial database. And we achieved most of the above tasks we set out to do.

This project highlights the potential for integrating cloud-based facial verification into video conferencing platforms, providing a scalable and secure solution to enhance participant authentication and meeting security. Future iterations can extend functionality to monitor user attentiveness and improve system efficiency.

## Related Work

Indla, Raghavendra Kumar, "An Overview on Amazon Rekognition Technology" (2021). Electronic Theses, Projects, and Dissertations. 1263. https://scholarworks.lib.csusb.edu/etd/1263

AWS Rekognition Documentation
https://docs.aws.amazon.com/rekognition/

## Templates we used during development:

AWS Amplify Template

https://aws.amazon.com/getting-started/hands-on/build-web-app-s3-lambda-api-gateway-dynamodb/module-one/?nc1=h_ls

AWS Face Liveliness Template

ZoomApp Template
https://developers.zoom.us/docs/zoom-apps/reference-apps/

https://github.com/zoom/zoomapps-sample-js

https://github.com/zoom/zoomapps-advancedsample-react