

# Attendance Capturer Analysis Report

## Group 6 TripleSix

Hanze Wang (39408695)

Isabella Yang (34096651)

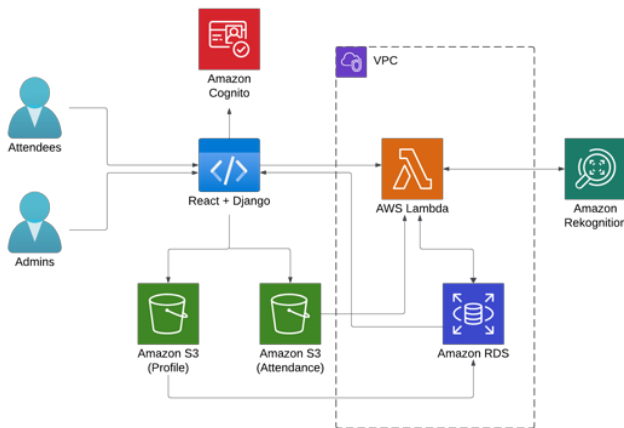
Joanna Shao (68441187)

Miranda Tang (13159264)

### Introduction

Our application is designed to tackle the hassle of manual attendance. Taking attendance manually is inefficient, error-prone and might lead to disruptions when required check-in tokens (e.g. ID, membership card) are unavailable. It is also hard to get quick and useful insights from paper records. With our application, attendance-making is faster, more accurate, and less disruptive, as it is a cloud-based automatic attendance system that does not require physical check-in tokens.

### Architecture



Our application supports two workflows. The admin workflow starts with secure authentication with Amazon Cognito. Once logged in, admins can add/edit attendee profile pictures through the React frontend which are uploaded to the profile S3 bucket and the RDS database. The admins can also query the RDS database for both profile and attendance data for visualization purposes.

For the second workflow, the attendees can use the React frontend to capture attendance photos and submit their profile IDs. The captured photos are uploaded to the attendance S3 bucket. Subsequently, the Lambda function is triggered with the URL of the attendance S3 object. It also queries the RDS database for the corresponding profile S3 object, then it makes an API call to Amazon Rekognition with both S3 objects to perform facial comparison. The result from Rekognition is returned to the Lambda function, which will insert the attendance data into the RDS database once facial comparison is confirmed to pass.

The Lambda function and RDS database are set up to be inside a VPC, and the Lambda function uses VPC endpoints

for accessing Rekognition and S3 services, which enhances the security of our application. Lastly, our application is fully deployed using Render.

### Explanation of Chosen Services/Tools

For cloud storage service, we chose to use AWS S3. It is suitable for our application as it is an object storage service that provides scalability, high availability, security, and performance. We are using two S3 buckets for storing profile and attendance related data respectively. Its high scalability and low latency ensure that our application runs smoothly even when there are high demands in the system (e.g. many people checking in around the same time). It also ties in well with the use of Rekognition as it takes S3 objects as inputs.

For computing component, we used AWS Lambda and Rekognition. Lambda offers a serverless environment for running code without the hassle of provisioning or managing servers. The cost of Lambda is based on the number of executions and compute time, making it highly cost-effective for our application's workloads. Lambda also integrates natively with services like S3, RDS, and Rekognition, allowing smooth workflows. For facial comparison, we used Rekognition as it is optimized for processing and comparing facial features. It allows for setting similarity thresholds which enables the application to define what constitutes a match; this flexibility allows us to fine-tune the facial comparison to the application's needs. It is also a pay-per-use service, which aligns well with our application's usage patterns where processing occurs only briefly after photo uploads.

For database, we decided to use RDS. The data that our application stores is inherently relational, where each attendance record is linked and has a structured relationship to the respective profile. RDS, being a relational database, is optimized for managing such structured, interdependent data. Our application also queries the database for visualizing attendance and profile data, which often requires complex operations like aggregations and filtering. RDS supports SQL queries, which are well-suited for such operations. We are also able to make sure our data is always up to date for visualization as RDS supports ACID transactions, meaning that it always ensures strong consistency.

These chosen services also fit well together in the context of a VPC. By having our Lambda and RDS in a VPC,

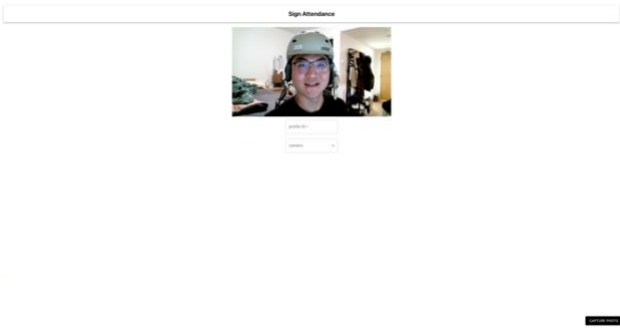
we have enhanced security as there is fine-grained control over inbound and outbound traffic through security groups. We also get low-latency and secure access to the database from our Lambda function. VPC endpoints were set up for both S3 and Rekognition, which eliminates exposure to the public internet as our Lambda accesses these services. Overall, these chosen services ensure that sensitive data is kept safe from public exposure while enabling secure, seamless interactions between services.

### Implementation Walkthrough

#### 1. Problem Scope and Design Decisions

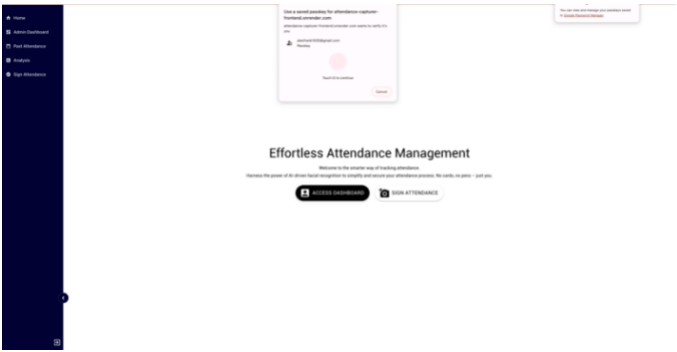
To address the problem of attendance tracking while ensuring scalability and security, we made several key design decisions:

##### A) Unique User ID for Profile Retrieval



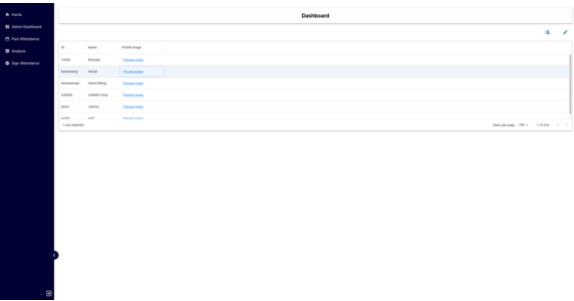
Each user is assigned a unique user ID, which they must input to retrieve their profile image. This simplifies the problem of user identification, reducing the complexity significantly. Without a unique ID, we would face a classification problem or require a minimum of  $O(n)$  operations to search through all user profiles. Such an approach would be computationally expensive and not scalable for large datasets. The use of unique IDs allows us to efficiently locate the user's profile with  $O(1)$  complexity using database indexing.

##### B) Second-Step Verification for Sensitive Information



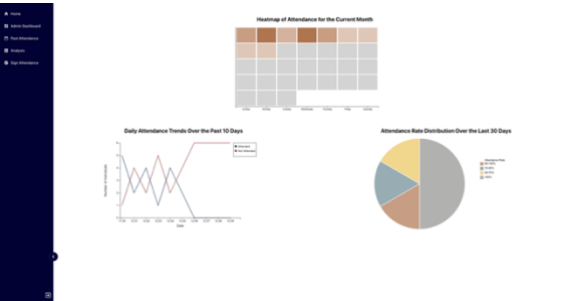
Considering the operational scenario where the admin's machine hosting the attendance system might be accessible to users during the signing process, we implemented a second layer of verification. This ensures that sensitive information, such as user data, dashboard statistics, and analytics, remains accessible only to administrators. This added security measure prevents unintended exposure of sensitive information to unauthorized users.

##### C) Role-Based Access Control with Admin Attributes



Since different administrators manage different activities, we included an admin attribute in the user profile data model. This attribute links users to their respective administrators. As a result: Administrators can only access the data of users under their management. This ensures that administrators have a clear view of their responsibilities without interference from unrelated activities.

##### D) Dashboard



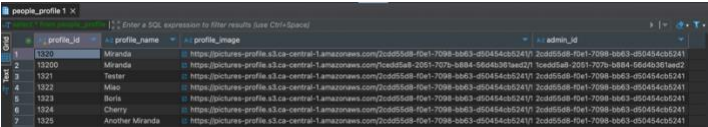
We designed a comprehensive data visualization dashboard to provide administrators with valuable insights into attendance trends and patterns. The dashboard aggregates data from the attendance records and presents it in an intuitive

format using visual elements such as charts and graphs. Administrators can analyze attendance rates over time, identify peak activity periods

2. Data Model Design

The system currently uses two tables to store and manage data:

A) User Profile Table

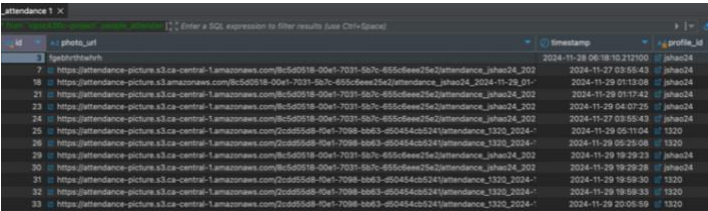


profile_id	profile_name	profile_image	admin_id
1200	Miranda	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	
1200	Miranda	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	
1201	Tefer	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	
1202	Mia	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	
1203	Boris	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	
1204	Cherry	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	
1205	Another Miranda	https://pictures-profile.s3.ca-central-1.amazonaws.com/2c0d55d8-f0e1-7098-b863-d50454c52417-2c0d55d8-f0e1-7098-b863-d50454c52417	

This table contains essential information about each user, including:

- user\_id: The unique identifier for each user.
- profile\_name: The name of the user.
- profile\_image: A link to the user's profile image stored in an S3 bucket.
- admin: The identifier of the admin managing this user.

B) Attendance Table



attendance_id	user_id	timestamp	image_url
1	1200	2024-11-28 06:16:10.212100	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024_11-28-06-16-10-212100
2	1200	2024-11-27 03:55:43	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-27-03-55-43
3	1200	2024-11-29 01:13:08	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-01-13-08
4	1200	2024-11-29 01:17:42	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-01-17-42
5	1200	2024-11-29 04:07:25	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-04-07-25
6	1200	2024-11-27 03:55:43	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-27-03-55-43
7	1200	2024-11-29 05:11:04	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-05-11-04
8	1200	2024-11-29 05:25:06	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-05-25-06
9	1200	2024-11-29 19:29:23	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-19-29-23
10	1200	2024-11-29 19:29:28	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-19-29-28
11	1200	2024-11-29 19:59:30	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-19-59-30
12	1200	2024-11-29 19:59:33	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-19-59-33
13	1200	2024-11-29 20:05:09	https://attendance-picture.s3.ca-central-1.amazonaws.com/8c5d5918-d0e1-7021-567b-855f8ee25a2attendance_image24_2024-11-29-20-05-09

This table logs attendance data for each user, with the following attributes:

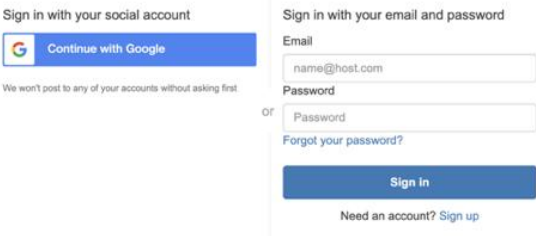
- attendance\_id: An auto-incrementing identifier for each attendance record.
- user\_id: The identifier linking the attendance record to the user.
- timestamp: The time when the attendance was recorded.
- image\_url: A link to the image captured during attendance stored in an S3 bucket.

By storing the image URLs in RDS (Relational Database Service), we enable easy querying with complex logic while leveraging S3 for scalable storage of the actual images. This separation of storage (S3) and indexing/querying (RDS) allows for efficient processing, especially when filtering or aggregating data across large datasets.

Security Features

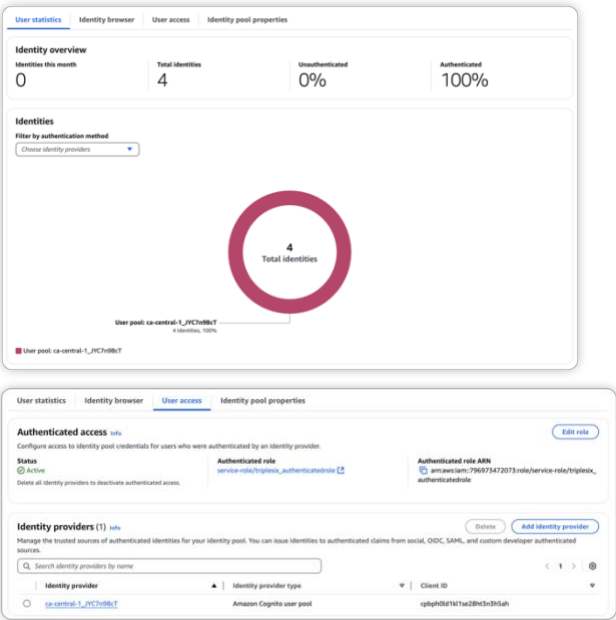
1. Primary Authentication

We use AWS Cognito User Pool as our primary authentication service. Users can either create an account using their email and password, or they can simply sign in with their Google account. Under the hood, this is implemented using industry-standard OAuth 2.0 and OpenID Connect protocols.



When users successfully authenticate, they receive ID, access and refresh tokens. The ID token is particularly important as it serves as proof of authentication to AWS Identity Pool.

Identity Pool verifies the token and in exchange, provides temporary AWS credentials. These credentials are then mapped to specific IAM roles, giving us fine-grained control over what AWS services each user can access. For example, in our S3 buckets for profile and attendance pictures, each admin can only access their own folder, identified by their unique Cognito identity. This ensures complete isolation of user data even within the same storage bucket.



**Permissions defined in this policy** [Info](#) Edit Summary JSON

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Q Search

Allow (3 of 436 services) Show remaining 433 services

Service	Access level	Resource	Request condition
<a href="#">Cognito Identity</a>	Limited: Read	All resources	None
<a href="#">Lambda</a>	Limited: Write	region string like [ca-central-1, FunctionName] string like [facialRecognition]	None
<a href="#">S3</a>	Limited: List, Permissions management, Read, Write, Tagging	Multiple	None

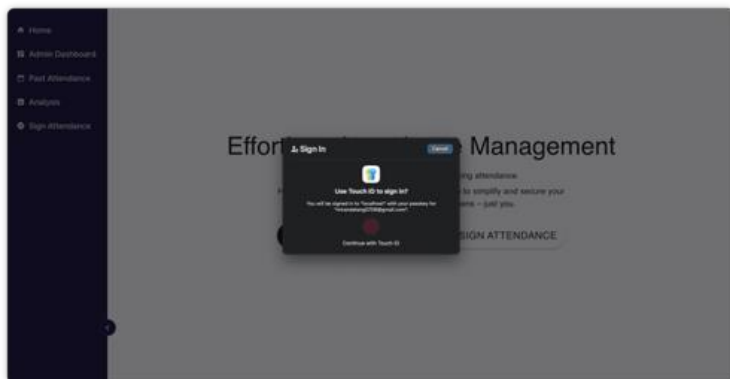
```
{
  "Sid": "AllowUserFolderCreation",
  "Effect": "Allow",
  "Action": [
    "s3:PutObject"
  ],
  "Resource": [
    "arn:aws:s3::pictures-profile/*",
    "arn:aws:s3::attendance-picture/*"
  ]
},
{
  "Sid": "AllowActionsInUserFolder",
  "Effect": "Allow",
  "Action": [
    "s3:*"
  ],
  "Resource": [
    "arn:aws:s3::pictures-profile/${cognito-identity.amazonaws.com:sub}/*",
    "arn:aws:s3::attendance-picture/${cognito-identity.amazonaws.com:sub}/*"
  ]
},
]
```

## 2. Multi-Layer Access Control

Building upon this foundational authentication system, we've implemented additional security layers to handle different user roles and access levels.

At the base level, we have unprotected routes like login page. Once users authenticate through Cognito, they gain access to first-layer protected features, allowing attendees to check in their attendance on admin's device without compromising security.

For admins managing user profiles and analyzing attendance data, we implement a second layer of security using TouchID or system password. This additional verification ensures that sensitive administrative functions remain protected.



## TouchID Registration

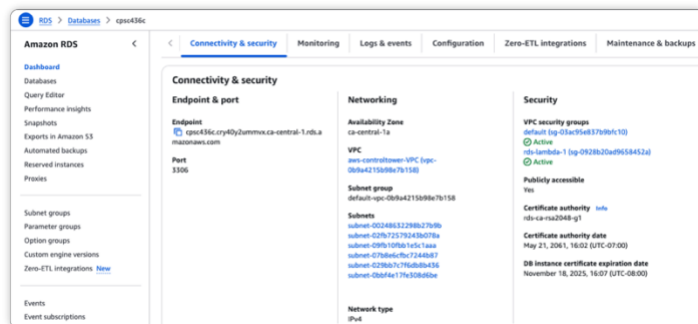
To access the dashboard, you need to register your device's TouchID. This ensures secure access to sensitive attendance information. Click the button below to start the registration process. You'll be prompted to use your device's TouchID or biometric authentication.

[REGISTER TOUCHID](#)

Re-verification is required when admins return to protected areas after visiting general pages like welcome or check-in. We also track the protected state across browser tabs and automatically clear it on tab closure, preventing unauthorized access through abandoned sessions.

## 3. Network Security Infrastructure

In addition to our authentication system, we must establish secure database connections. Our database is hosted within a VPC, and we use security groups as our main method for managing these connections. The security rules are specifically configured to permit MySQL traffic through port 3306, with access restricted to authorized IPs.



## 4. Input Sanitization

Finally, to prevent injection attacks, all input fields have character limits and only accept letters and numbers.

**Add New Profile**

1 — 2

Enter Profile ID And/Or Name      Upload Profile

Profile ID \*

/

Profile ID must be alphanumeric and less than 60 characters.

Profile Name \*

isabellatestteststdfg

BACK      NEXT

**Add New Profile**

1 — 2

Enter Profile ID And/Or Name      Upload Profile

Profile ID \*

abcs

Profile Name \*

/

Profile Name must be alphanumeric and less than 60 characters.

BACK      NEXT

## Cost Analysis & Prediction

Based on our projected usage of 200 users making daily check-ins, we've calculated the monthly costs across our AWS services.

The primary cost driver is our RDS database, running on a t4g.micro instance at around \$21.30 per month, including storage. For facial recognition, AWS Rekognition will process about 6,000 comparisons monthly, costing \$6. Our S3 storage needs for both profile and attendance images are minimal, totaling just \$0.07 monthly for roughly 1.55 GB of storage and associated requests.

This brings our total monthly operating cost to around \$27.37. Looking at scalability, if our user base doubles to 400 users, our costs would increase modestly to about \$33.44 monthly, with the main increase coming from additional Rekognition calls.



For future cost optimization, we plan to implement image compression to reduce storage and transfer costs, while continuously monitoring usage patterns to adjust our resources.

## Performance

Our application integrates a series of design optimizations aimed at enhancing performance, scalability, and user experience. Below, we discuss the strategies employed and their impact on the system's efficiency:

### 1. Optimized API Calls

Each webpage is designed to perform as few API calls as necessary, retrieving bulk data efficiently. This reduces the number of network requests and improves page load times. In addition, we've enhanced our data retrieving and processing flow by utilizing some centralized processor. The centralized processors uniformly pre-clean data before routing it to the appropriate handlers. This ensures consistency, reduces duplicate effort, and optimizes subsequent fine-grained extraction and transformation tasks. By handling data transformations closer to where they are consumed, we achieve faster analysis and computation, presenting results to users in near real-time.

### 2. Authentication Management

We integrated cognition into our authentication workflow aiming for a scalable secure and robust solution. Upon successful login, Cognito issues JSON Web Tokens. And these tokens are securely stored in the browser's localStorage for quick access. Therefore, we reduce the need for repeated server-side validation during navigation. Upon logout, we will remove the tokens from local storage to ensure no residual authentication in client side and will trigger the cognito-side to end the session.

### 3. Scalable image storage

Instead of storing image pixels directly, the database only holds references (URLs) to the images stored in Amazon S3. This approach will minimize the size and complexity of database as URLs only takes a tiny fraction of the space need to store image binaries. This also ensures quick and efficient data retrieval and joining tables. In addition, with a smaller-sized database, our database replication will be smaller as well, hence this will reduce the cost for backup. Furthermore, S3 is highly scalable and designed to handle vast amounts of data. It allows us to store virtually unlimited images with consistent performance as it is optimized for serving static assets. Nevertheless, compared with traditional solution, S3 has lower storage costs yet higher data durability. And S3



automatic replicate across multiple regions to achieve built-in redundancy and high availability.

Overall, in our deployed environment, we have an optimal performance where uploading a standard picture takes about 6s. The time to load the past attendance is around 1s for 10 users. And it takes about 5s to generate three analytic graphics for an attendance table with 80 records. Our local version has faster performance since we used free version of the deployment platform.

### Challenges and Lessons Learned

One of the major challenges we faced was integrating technical components that were unfamiliar to us. For instance, while we were accustomed to using FaceID for unlocking phones, our group lacked prior development experience with facial recognition technology. As a result, the mechanics of facial comparison initially felt like a "black box." During the design phase, we conducted pilot research on various facial recognition providers, such as AWS Rekognition, to assess their feasibility for our application. We realized the crucial importance of having a solid design early on. Although we went through several drafts before arriving at a feasible and satisfying workflow, this iterative process ultimately saved us significant time and effort. By clarifying potential points of confusion and addressing bottlenecks during the design phase, we were able to streamline development and avoid major setbacks later. After finalizing the design structure, we delved into learning the fundamentals of AWS Cognito and Rekognition. Nevertheless, this led to a new challenge: integrating these services with others, such as S3 buckets and AWS RDS, to build a seamless application.

To address this, we self-learned Cognito, implemented the login process, and stored authentication tokens locally on the client side. Next, we tackled the distinction between public and privileged routes, ensuring admin-only pages were secured using token-based authentication. Another significant challenge involved designing an AWS Lambda function to retrieve images from two distinct S3 buckets, pass them to Rekognition for facial comparison, return the results to the client side, and store them in AWS RDS. This Lambda function became the core of our application, facilitating cross-service communication. However, debugging the Lambda function proved difficult, as errors often occurred intermittently or were rooted in misconfigurations. In the end, we added a VPC endpoint and configured inbound rules to allow communication with Rekognition. Through this process, we learned the importance of building and testing incrementally. Isolating two components at a time made identifying and resolving issues far more efficient than sifting through the entire system.

Lastly, ensuring Rekognition's accuracy posed its own set of challenges, particularly in minimizing false positives and false negatives. To evaluate its performance, we designed extensive test cases involving partial facial obstructions like hats, masks, glasses, and gloves. Even in extreme cases, such as a combination of a hat covering the forehead and a mask covering the mouth and nose, Rekognition successfully distinguished between positive and negative matches. Further testing with relatives demonstrated that Rekognition could accurately differentiate faces despite familial similarities. Based on these tests, we configured Rekognition's confidence threshold to 80, striking a balance that proved especially valuable for recognizing masked faces in a post-pandemic context. The key lesson we learned was the importance of tuning parameters to align the application's behavior with its goals. By iteratively testing various scenarios, we reduced false positives and negatives, thereby refining the system's overall accuracy.

As future improvements, we would like to add some features to enhance user experience. First is we do realize the process of adding user can be cumbersome and tedious in the case of creating many users at once. Therefore, we would like to reduce the hassle by providing the feature of bulk creation to ease the user's workload. Furthermore, as users may have attendance data already existing on some other platform. We would like to simplify the migration process as well by adding the feature of supporting data import through csv files. In addition, our current default time unit of an event is one day. We would like to relax this limitation by adding an object type of event which defines the start time and end time so users can have events running from a couple hours to even several days.

### Links

GitHub repository:

<https://github.com/Miranda-Tang/attendance-capturer>

Application:

<https://attendance-capturer.onrender.com>

We deployed our application on Render and we're using the free version to minimize cost, thus it may take a while to load the pages.