ECE1779 Assignment 3 Report

Group #5

Yuhan Wei 1000893161
Hsuan-Ling Chen 1002322202
Eric Wang 1002294108

1.0 Introduction

The application designed for Assignment 3 aims to be cool and captivating. It builds on the photo app from assignment 2 and includes new features such as AWS location, Rekognition, and a puzzle game. While it also shows cases of the usage of AWS services such as Lambda functions, API Gateway, and DynamoDB.

2.0 Application Description

Introducing our latest web application! Built on the foundation of Lambda functions, API Gateway and DynamoDB, and packed with new cool features. This latest application lets you add your locations on the map, tag and search for images, and a fun puzzle game using your uploaded images. Whether you're a photographer, traveler, or just looking for a fun way to pass the time, our web application offers a unique, engaging experience, and easy-to-use interface; sharing your adventures with the world has never been easier! Start your adventure today!

How to use the Application:
On the home page, users can first upload an image by clicking on the ‘Upload Image’ button and fill out the relevant image information including the image, image key, date, location, tag and description. Once completed, the user can upload this information to the server and is able to use this newly uploaded image to search using tags and display the image. The two other newly added tabs ‘Have Fun’ and ‘Track your memory’ contain this application's new features.

2.1 Puzzle Game

The 'Have Fun' tab brings you to the game page, where the user can select the image and game difficulty level, and the image will turn into a puzzle for the user to solve. The idea behind this system was to offer an enjoyable and engaging platform for users to play personalized puzzle games with their own images. Implementing such a system came with several challenges, such as ensuring optimal game performance, creating a user-friendly interface, and developing an efficient leaderboard management system. In this application, we have utilized Amazon API Gateway, Quicksight and S3 to complete the Leaderboard Dashboard and Storage. This enables the user to build an intimate relationship with their own images.

Figure 2.1: Puzzle Game Interface

2.2 AWS Location Service

‘Track your memory’ allows users to visualize and keep track of the geolocation of their uploaded images on a map. The system draws on the AWS Location Service. The idea behind this feature was to extract the exact longitude and latitude of where the image was taken from the image's EXIF information. However, this information is often removed online for privacy purposes, making it challenging to obtain. The implementation challenge was addressed by using AWS Location Service geocoding. It makes this task easier by marking the location on the map with the location information the image uploaded. During the design process, the marker is later upgraded as the image and displays the image information on the map. For the design decision, the place index requires authentication for location service, but we wanted to allow everyone to have access to the map. Hence, Amazon Cognito is also used to grant anonymous access to the application, which allows you to create a richer authorization with a policy to define what can be done by unauthenticated users.

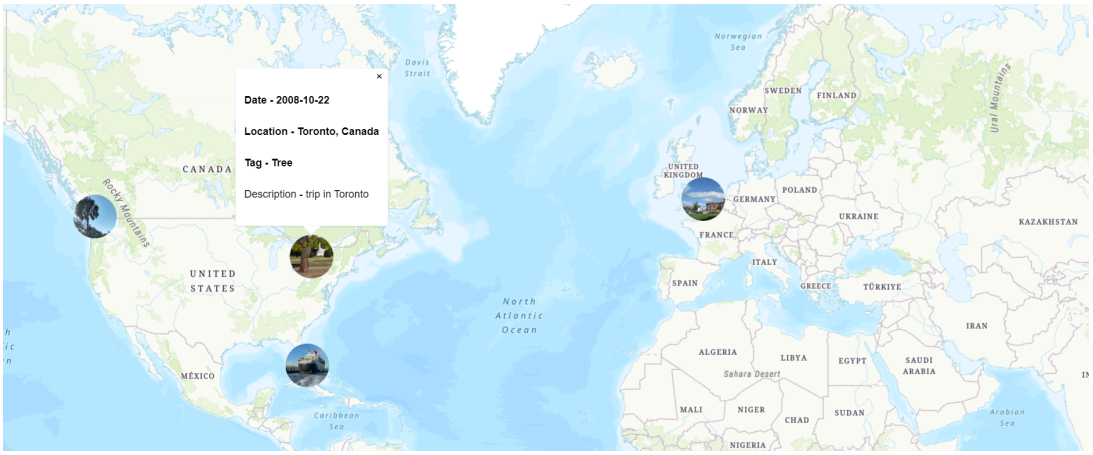


Figure 2.2: Location Service Interface

2.3 AWS Rekognition

In the application, the user is now able to search for images using tags, and in order to simplify this step in practice, we provided a tag generator to help the user come up with tags. This generator uses AWS Rekognition to quickly analyze the image and provide suggestions to the user. The idea behind this feature was to offer users an alternative way of searching for images when they could not find the image key value immediately.

Implementing this feature came with several challenges, such as making sure that the AWS Rekognition service accurately suggests relevant tags, and ensuring the feature is user-friendly and intuitive on the user interface. These challenges have been successfully overcame by the thoughtful frontend design, shown in Figure 2.3, allowing users to easily search for images using tags.

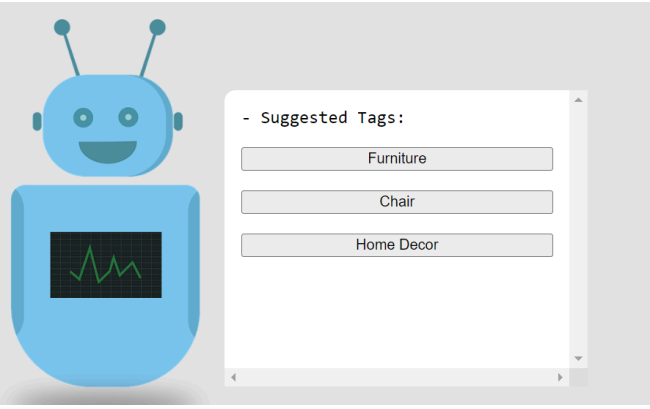


Figure 2.3: Image Rekognition Tagging Interface

3.0 Application Architecture

The main web application, such as the serving client requests, stayed the same from Assignment 2. In this assignment, the application includes a separate process implemented as AWS Lambda functions run in the background. In our application, four main lambda functions are called for the purpose of Athena DynamoDB Connector, Album Dataset Refresh, New Record Update and Game Dataset Refresh.

3.1 Background Process

The background process is implemented using AWS services. In the DynamoDB where all the photo information storage is read, the statistics are visualized in the Amazon QuickSight. [1] However, to read the data from the DynamoDB, it needs to go through the Amazon Athena to be displayed in QuickSight. Hence, this process of connecting Athena and the DynamoDB is placed into a Lambda function. Similarly, each time the DynamoDB is updated, it triggers another lambda function to refresh the data in the statistics dashboard.

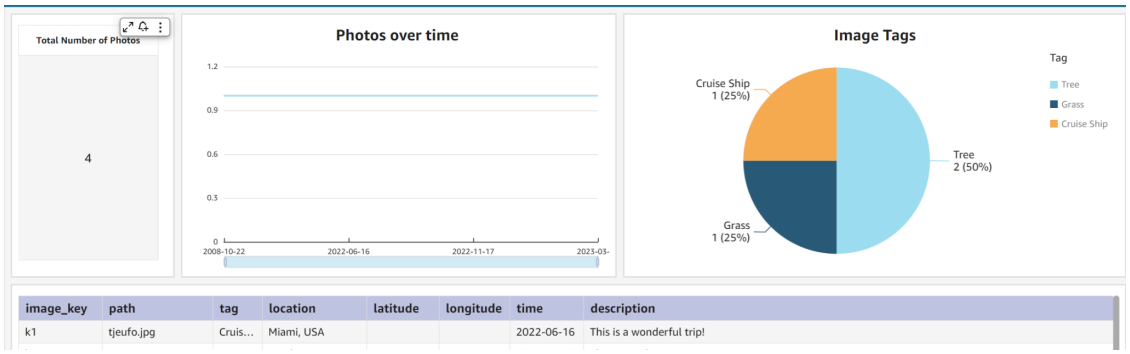


Figure 3.1 Album Statistic Dashboard

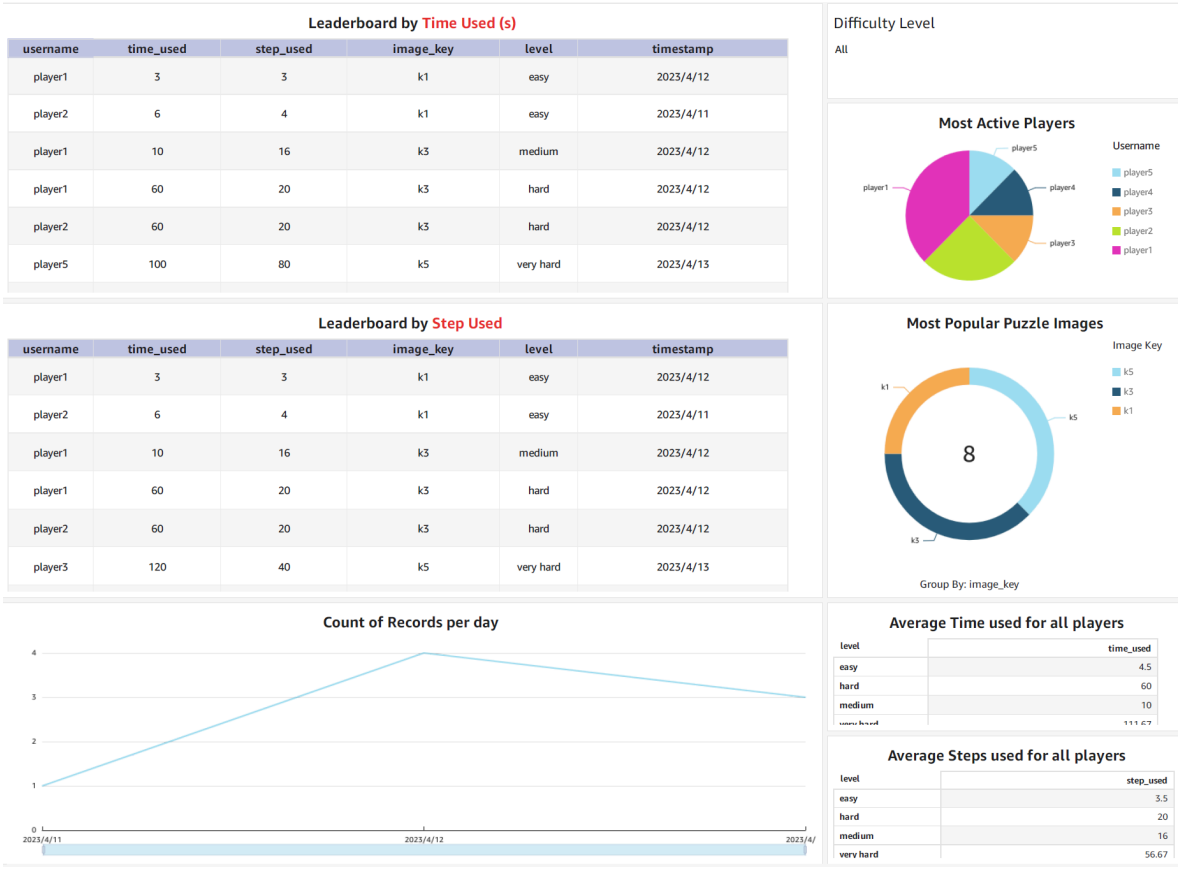


Figure 3.2 Game Leaderboard Dashboard

Similarly, the setup for the leaderboard dashboard is visualized in Amazon QuickSight. When a new record request is being made from the Puzzle Game front end, it goes through the Amazon API Gateway through HTTP Access, which triggers the AWS Lambda function to update the S3 storage. Once the new record is updated and stored in the S3 storage, another lambda function triggers to refresh the game dataset. The updated results can be visualized on Amazon QuickSight to display the Game Leaderboard Dashboard. The overall application architecture can be seen in Figure 3.3 below.

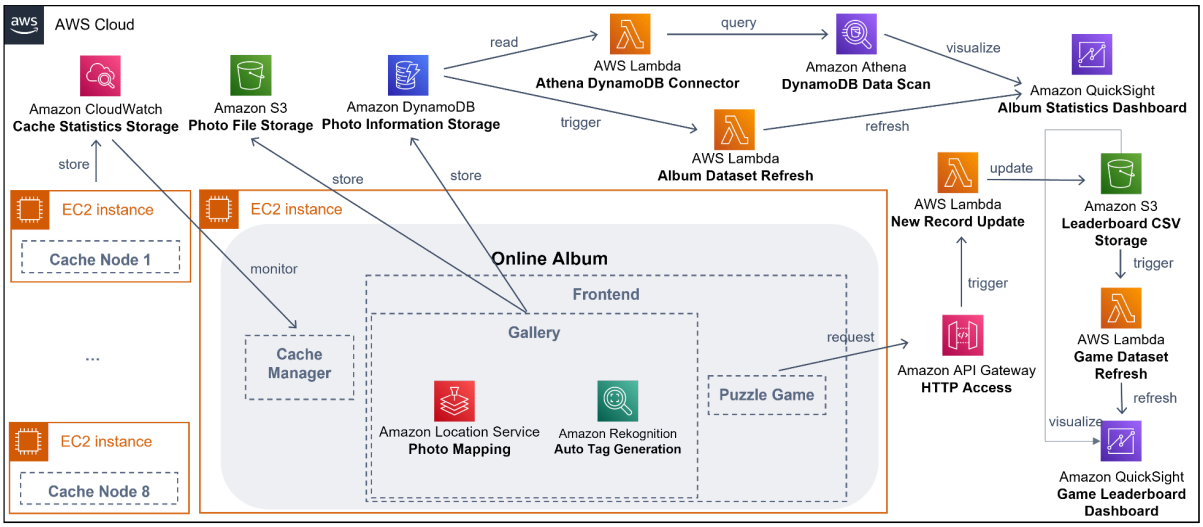


Figure 3.3. Architecture of the Application

4.0 Cost Model

To better understand the usage of different AWS services in the current application configuration, we conducted a cost projection. We estimated usage based on 10-user cases and scaled up linearly for 1,000 and 1,000,000 users. The application's core services include Lambda, DynamoDB, S3, and EC2, and the following assumptions were made: The underlying assumption is that the users would treat the app as a virtual album where only selected photos are uploaded, instead of a camera roll where photos are uploaded as taken.

We assumed that users would treat the app as a virtual album and only upload selected photos, not all photos taken like a camera roll. Therefore, we estimated that users would upload between 10-100 photos daily and use the app 1-10 times daily. Each photo's size ranges from 100 KB to 10MB, and each database entry is about 1KB. Additionally, we assumed that the app would use 8 cache nodes, and the memory allocated would increase logarithmically.

We used the AWS Pricing Calculator to calculate the estimate, and the detailed inputs are included in Appendix B. The following table 4.1 and figures 4.1 and 4.2 provide a summary of the expected amount and percentage breakdown by services.

Table 4.1 Summary of Application Cost Projection

	10 Users	1k Users	1M Users
6mo Sum	6,399	17,223	10,500,000
Upfront	180.0	180.0	165,000

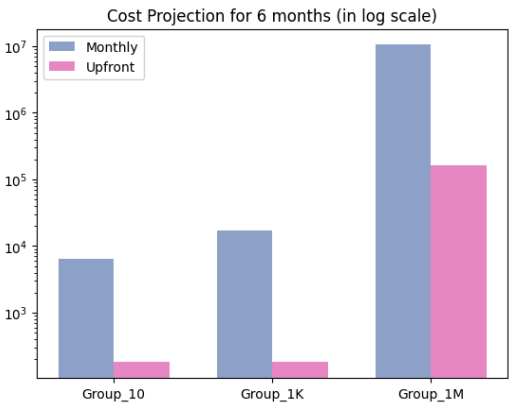


Figure.1 Cost Projection for 6 Months

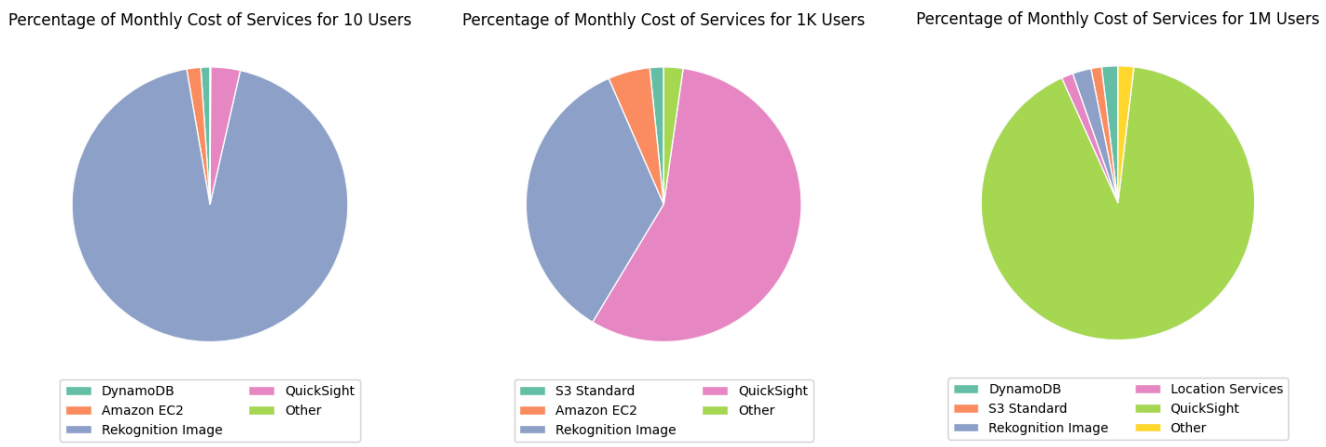


Figure 4.2. Percentage of Monthly Cost by Services

5.0 Evaluation Results

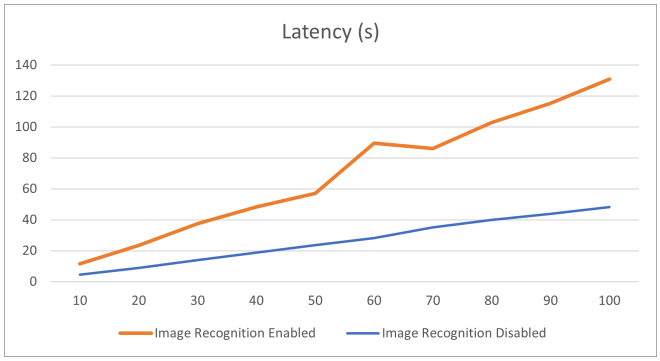
We invoked AWS Image Recognition service when uploading the photo, which introduced some delay in the application. Therefore, we decided to evaluate the latency throughout with and without image recognition functionality to measure the performance of our application.

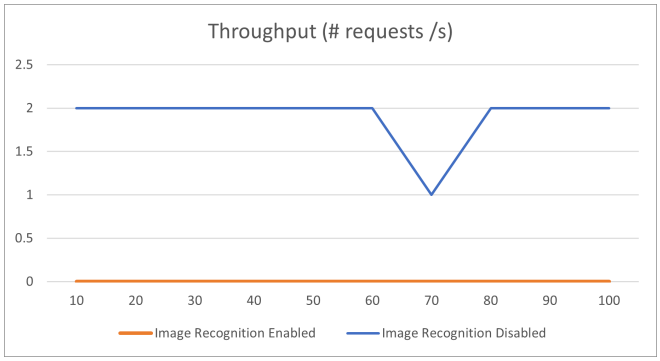
5.1 Evaluation Setup

- Set the cache capacity to 10 MB and 1 cache node. .
- Prepare 12 test images with a size range from 5 KB to 2 MB.
- Design 10 writing request batches from 10 to 100 with a step of 10.
- For the writing operation, automatically generate keys such as ‘test1’, and ‘test2’ and randomly select images from 12 candidate images.
- Measure latency and throughput for each request batch.
- Run the evaluation with and without image recognition functionality after our application is deployed on AWS.

5.2 Performance Graphs

As we can see in the following two charts, the image recognition service introduces a large latency into the system. As a result, the system can not finish one uploading request within 1 second. We think the performance of the AWS image recognition service depends on the image as well. We would expect low latency on the image with the obvious objects. Based on these results, we plan to add a new feature to allow the user to enable or disable the image recognition service.





References

[1] Leelapornudom, N. (2022) *Visualising your Amazon dynamodb data with Amazon QuickSight*, *DEV Community*. DEV Community. Available at: https://dev.to/awscommunity-asean/visualising-your-amazon-dynamodb-data-with-amazon-quicksight-14n4#step4 (Accessed: April 16, 2023).

Appendix A: Attribution Table

Team Members	Tasks
Eric Wang	● Image tag generation with Rekognition ● Cost estimation with AWS Pricing Calculator
Yuhan Wei	● Implemented puzzle game ● Designed and implemented the background process ● Integrated aws Rekognition, aws location service into frontend
Hsuan Chen	● Location service ● Cognito ● Report Writing

Appendix B: Cost Estimates with AWS Pricing Calculator