ECE1779: Introduction to Cloud Computer

Winter 2021

Assignment 3

Serverless Computing

Fulin Huang 1002942756

Ya Luan 998514433

Yilun Wan 1002674199

1. Introduction

In this assignment, the group designed and developed a serverless web application named **Bird's Eye**. The application provides an online platform for professional and amateur photographers to share their works with each other. During the Covid-19 pandemic along with social distancing, photographers are not able to hold photography exhibitions. We provide this platform for them to share their works with the public and to connect with other professionals.

The application and its background processes are all deployed through AWS Lambda while all the data and files are stored in Amazon DynamoDB and Amazon Simple Storage Service (S3) respectively. To provide better services to potential users, our application is integrated with Amazon Rekognition, allowing them to search for a particular category of images to explore. Moreover, a scheduled background process is implemented to recommend the most popular photos to the community.

2 User Manual

This section explains the functionalities of the application as well as how to use them.

Cover Page: This is an initial, visually-led page to welcome the users for using the Bird's Eye app, which acts like a teaser before launching the full website. It contains the sidebar for basic navigation and the footer for social media links of the app.

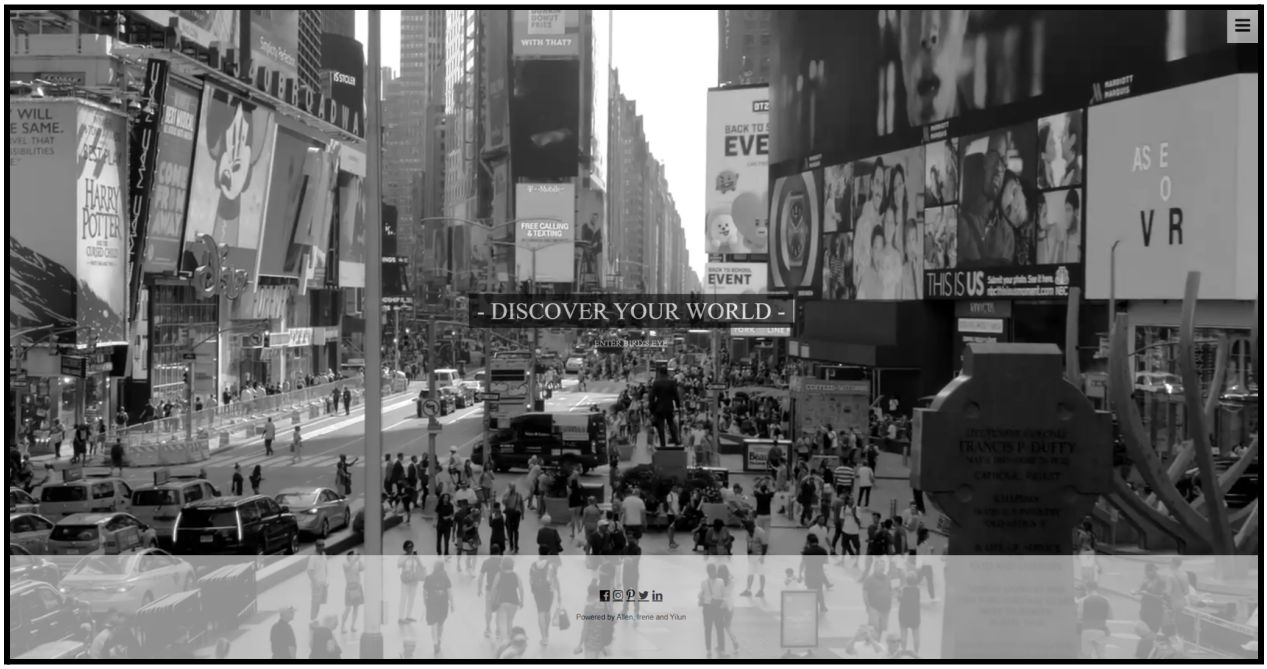

Figure 1: Welcome page

Home Page: Display all photos that the user uploaded in the order of uploading time (from old to new). It also shows the number of posts and the popularity of the user. Popularity means the total number of click rates of all his posts.

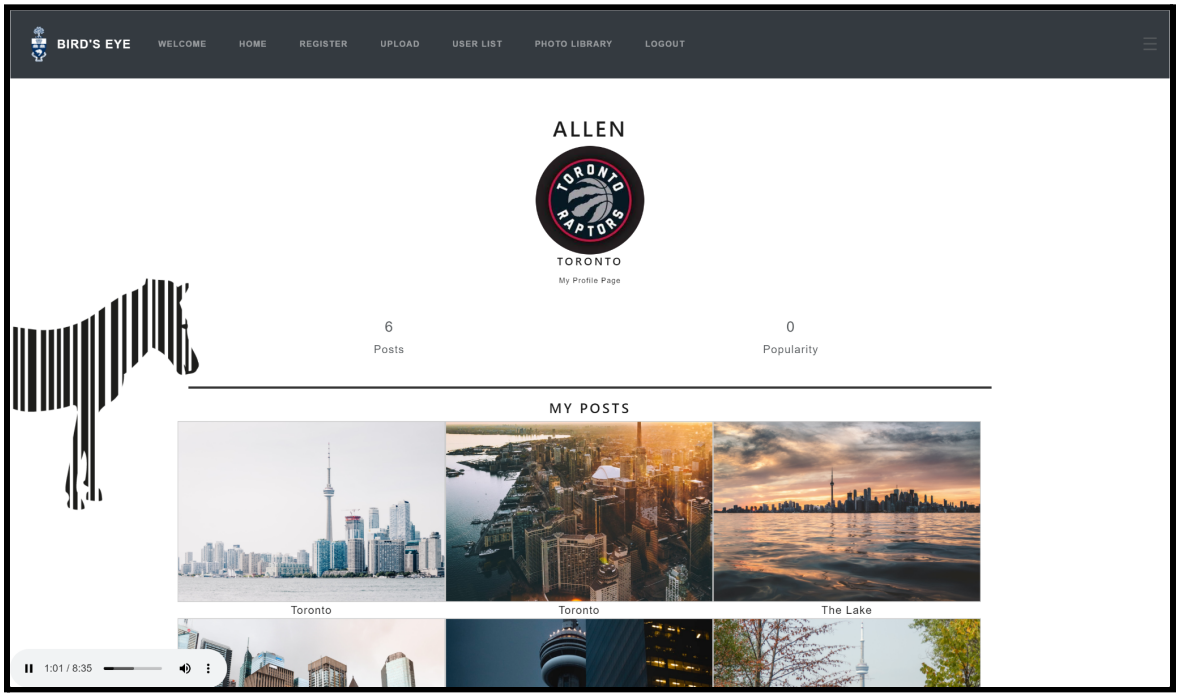

Figure 2: Home page

Register Page: Allow a photographer to register an account.

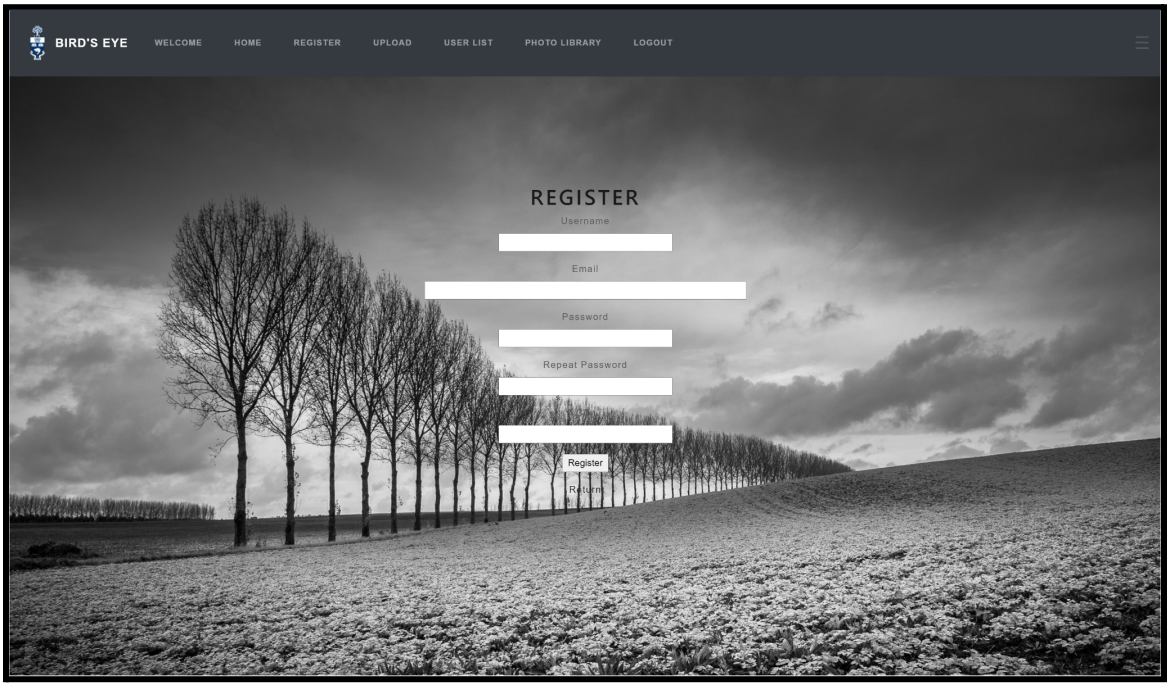

Figure 3: Register page

Upload Page: Allow the user to upload an image. Users can add an image title as well as a description to the image. The Amazon Rekognition will automatically generate a hashtag for the image. Users can modify the hashtags if needed.

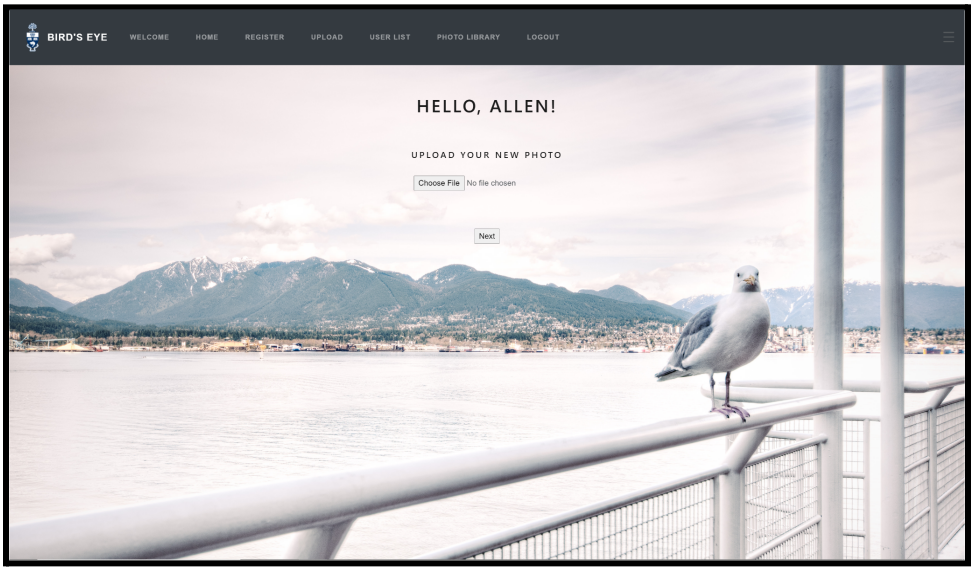

Figure 4: Upload page

Figure 5: Upload page 2

Login Page: Allow users to login.

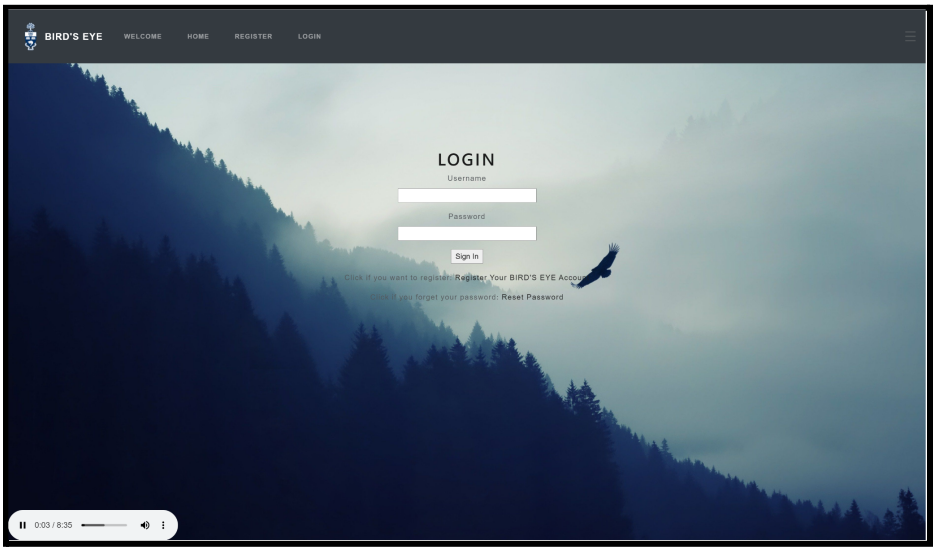

Figure 6: Login page

Reset Password Page: Allow users to reset passwords via emails.

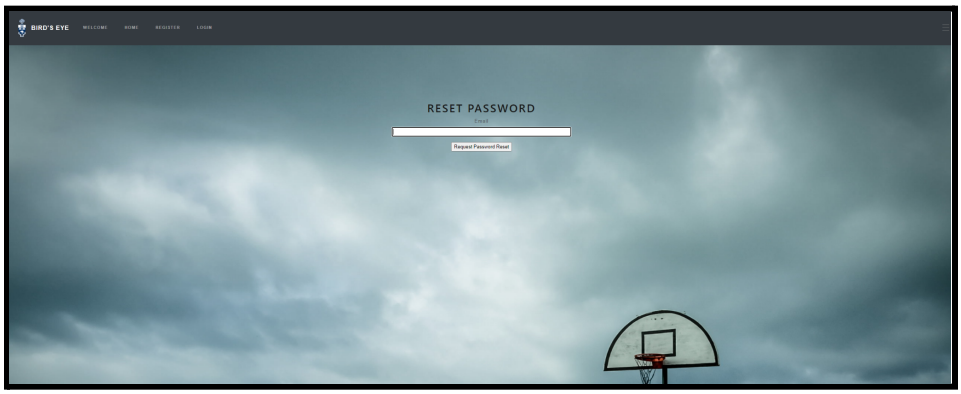

Figure 7: Reset password page

Change Password Page: Allow users to change their passwords.

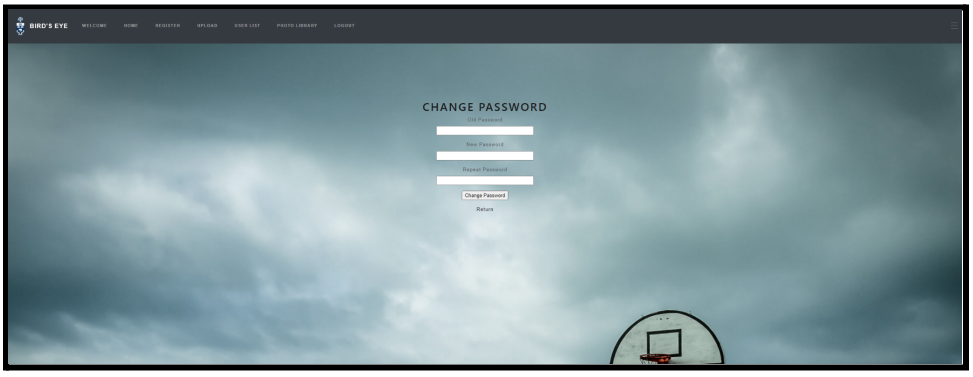

Figure 8: Change password page

User List: Display a list of users. Users can click on a user to view his/her posts.

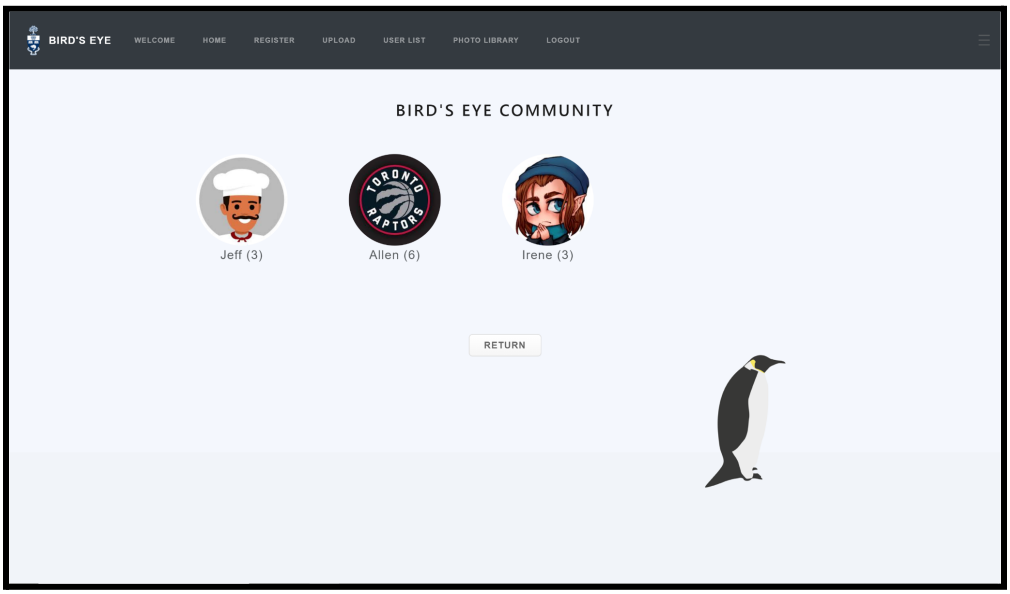

Figure 9: User list page

Photo Library: Display three recommended images at the top of the page for the user, and display images of all users in the order of uploading time to allow photographers to explore. If the user inputs a category in the search field, this page will display all images in this category. Users can click to view detailed information about the photo.

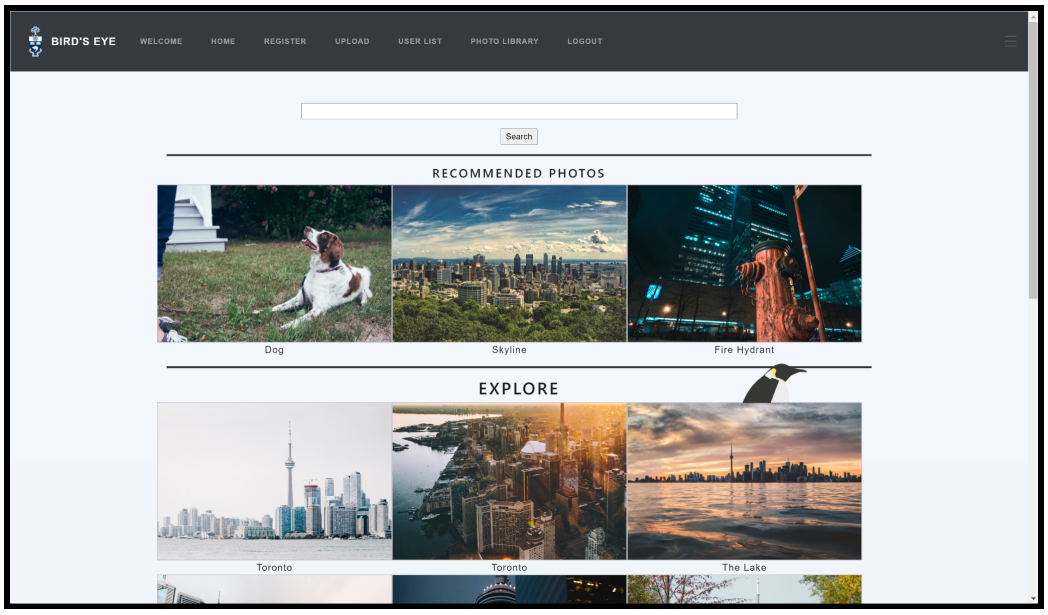

Figure 10: Photo Library page

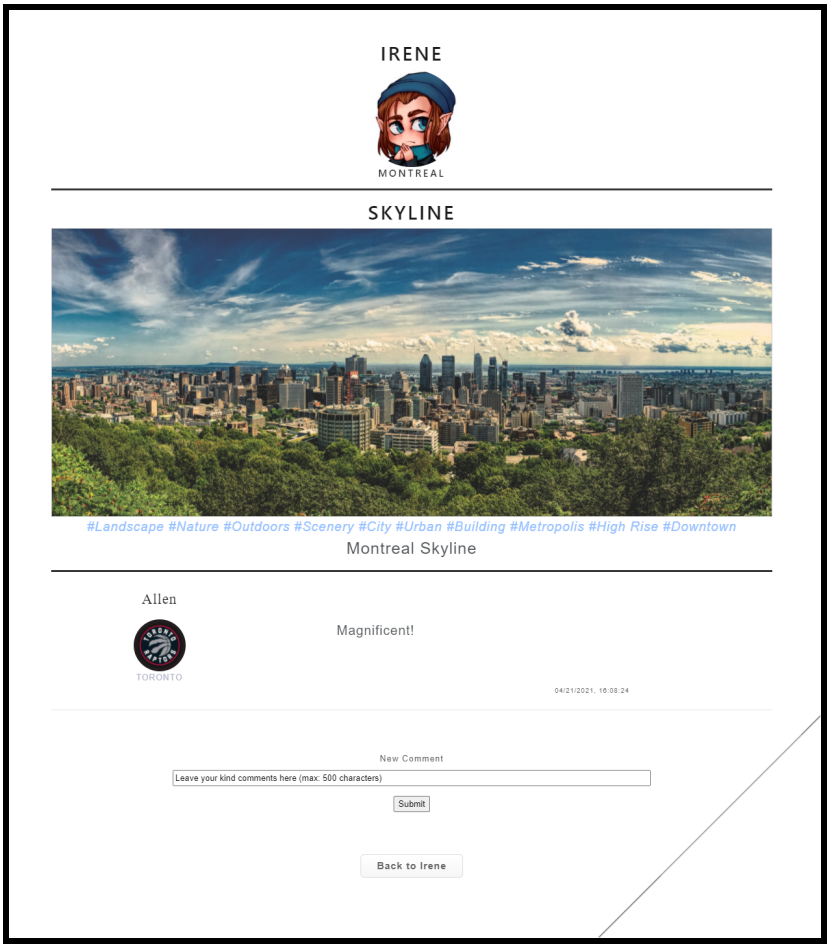

Figure 11: Post page

3 Application Architecture

3.1 Code Structure

The code structure is shown in Figure 12 where *main.py* is the launcher of the web application. The “venv” folder contains all the python packages that are required to run the application. The web application codes are written and packaged in the “app” folder. All the python scripts are further explained in **Appendix A**. Inside the “/app” path, the “templates” folder contains all the html templates.

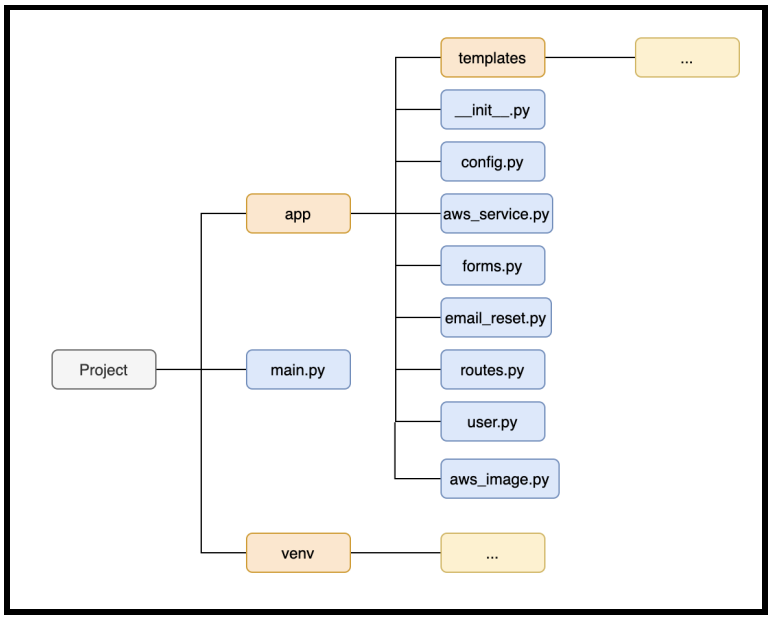

Figure 12: Code structure

3.2 Database and File Storage

All persisted data are stored in Amazon DynamoDB and S3. There are two tables in the Amazon DynamoDB -- user table and photo table, as shown in Figure 13 and Figure 14. The user table stores attributes that are related to the user's profile, and the photo table includes attributes that describe the photo itself. All photos in the application are stored in S3. There are two folders in S3 -- the avatar folder and the photo folder.

DynamoDB Table

User

- *Username*: username of the user
- *Password*: encoded password of the user
- *Email*: email for user to login their account
- *Address*: home/work address of the user
- *Self-description*: summary of user's work, interest, hobby etc.
- *Registration time*: date when the user registered the account
- *Last login:* date and time when user last login
- *Number of posts*: total number of posts that the user has uploaded
- *Active level:* active level of the user (depends on the # of posts per month)

Photo

- Username: username of the user
- Photourl: url of the photo
- *Hashtag* a list of hashtag associated with the photo
- *Photoname* title of the photo

- *Description* short description of the photo
- *Comments* a list of comments post by other users
- *Post time* date/time when the post is first available
- *Click rate* total number of clicks/view of a photo by other users
- *Is recommended* whether the photo is recommended (depends on the click rate).

S3

Avatar folder

Store avatars of all users.

Photo folder

Store posts that the users have uploaded.

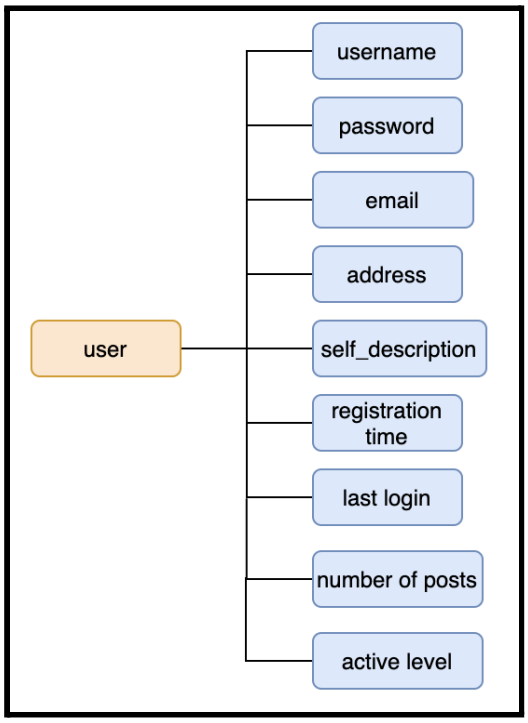

Figure 13: User Table

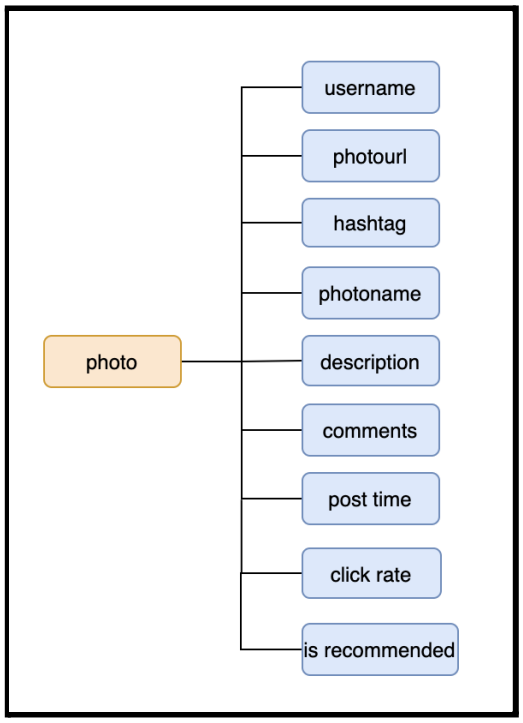

Figure 14: Photo Table

3.3 Background Process

In this web application, there is a background process which runs independently of the main application. It selects the three most popular photos based on the weighted click rate of each photo and how long the photo has been posted. The three selected photos are recommended to all the users on top of the photo library page. The background process is deployed as a lambda function and triggered by the html requests through AWS API Gateway from an AWS EC2 instance twelve times a day. That is, the recommended photos are refreshed twelve times a day.

In details, when the background process is triggered, it first updates the “active level” of each user. A user is considered to be active when he/she has registered for a long time and has

posted a decent number of photos. The “active level” will be used as the weight of each user’s click when calculating the click rate. Afterwards, the background process calculates the weighted click rate of each photo and the three photos with the highest weighted click rate are selected and recommended to the community. The equation of active level and weighted click rate are shown in Figure 15 and Figure 16.

$$active\ level = \max\left(1, \frac{current\ time - registration\ time}{30}\right) * 5 + \max\left(1, \frac{number\ of\ posts}{30}\right) * 5$$

$Where,$
$current\ time: Current\ date$
$registration\ time: Date\ when\ user\ first\ registered\ the\ account$
$number\ of\ posts: Total\ number\ of\ photos\ posted\ by\ the\ user$
$active\ level: weight\ of\ each\ user's\ click$

Figure 15: Active level equation

$$Weighted\ click\ rate = \frac{click\ rate}{current\ time - post\ time}$$

$click\ rate: Number\ of\ clicks\ or\ views\ for\ each\ post$
$current\ time: current\ day$
$post\ time: post\ day$

Figure 16: Weighted click rate

4 Cost Model

In order to estimate the operational cost of this serverless website, the group has created a cost model with some assumptions. The assumptions are listed below and the numerical results are summarized in Table 1. AWS permanent free tier services are taken into account.

User Behaviour Assumptions:

- Average request rate: 50 per user per day
- Average posting rate: 1 photo per user per day
- Average photo size: 2 mb per photo

Operational Assumptions:

- Average database item size: 3 kb
- Number of DynamoDB RCU and WCU: about 0.16 times the total number of users
- Average lambda function runtime: 500 ms
- Average background process runtime: 2000 ms

- Lambda memory usage: less than 128 mb
- User data backups: all data are backed up with the cheapest option
- Background process trigger: the cost is ignored.

	10 Users	**1000 Users**	**1000,000 Users**
AWS Lambda (Monthly)	0	0	1681.1695
AWS DynamoDB (Monthly)	0.00008869171143	60.84886917	72553.0921
AWS S3 (Monthly)	0.0134765625	30	1340.2625
Amazon Rekognition (Monthly)	0.3	30	20200
Total (Monthly)	**0.3135652542**	**120.85**	**95774.5241**
Total (6 Months)	**1.881391525**	**725.09**	**574647.1446**

Table 1: Cost Prediction (USD)

Appendix A: Code Documentation

routes.py

This file contains all the routes of this web application.

FUNCTION
def initial():
Redirect to initial.html to display the front page of the application.

def about_us():
Redirect to about_us.html to display information about the developers.

def home():
Display images that the user uploaded. Users can open a photo to view detailed information, such as hashtag and description.

def photos(hashtag):
Display a list of images under a category that the user entered in the search blank.

def photos_search():
Return recommended images and all images of all users for the user to explore.

def login():
Validate user inputs and redirect to the home page if login successfully. Otherwise, redirect to the login page.

def register():
Allow a new user to register an account. If registration is successful, it will redirect to the login page for the user to login.

def reset_password_request():
Allow the user to reset his password by following instructions sent as an email. If the action is successful, it will redirect to the login page for the user to login.

def reset_password(token):
Store new passwords in the database.

def change_password():
Allow users to change passwords. If the action is successful, it will redirect to the login page for the user to login.

def logout():
Logout the user. Redirect to login page.

def upload_page():
Allow users to upload a photo. If the action is successful, it will redirect to the next page for the user to enter more information about the photo.

def next_page():
Amazon Rekognition generates hashtags for the photo. User adds title and description to the photo. All information is saved in the Amazon DynamoDB.

def upload():
Redirect to upload page.

def change_profile():
Allow users to change profile information.

def upload_avatar_page():
Allow users to upload an avatar.

def upload_avatar():
Redirect to upload_avatar.html.

def user(username):
Allow users to view other user's posts.

def post(username, post):
Allow users to click on a specific post to view detailed information. Click rate increases by one.

def delete_photo(username, post):
Allow users to delete their own post. If the action is successful, it will redirect to his/her own page.

def list():
Display a list of users that are on this application.

aws_image.py

This file contains a function that generates a list of hashtags for each photo using the Amazon Rekognition.

FUNCTION
def detect_labels(photo, bucket):
Detect objects in an image using the Amazon Rekognition and return a list of hashtags.

aws_service.py

This file contains functions that manage S3 and Amazon dynamoDB.

FUNCTION
def create_user_table():
Create Amazon DynamoDB user table.

def create_photo_table():
Create Amazon DynamoDB photo table.

def create_s3_bucket():
Create a S3 bucket with a unique bucket name.

def add_table_item(table_name, item):
Add an item to the given table.

def update_table_item(table_name, key, column, new_value):
Update value of an attribute of a given table.

def increase_table_column(table_name, key, column, increase):
Add a column in the table to count the number of posts and click rate.

def delete_table_item():

Delete an item in the table.

def query_table_item(table_name, key, value):
Query the table given a partition key and a value.

def scan_table_item(table_name, key, value):
Scan the table given a key and a value.

def scan_table_contain_item(table_name, key, value):
Scan the table given a key and return responses that contain value.

def scan_table_condition_item(table_name, key, value):
Scan the table given a key and return responses that are larger than the given value.

config.py

This file contains all constant variables used in the application, such as S3 bucket name and allowed photo extension.

email_reset.py

This file contains all functions to send an email and reset the password.

FUNCTION
def send_password_reset_email():
A method that sends an email to reset the password.

def send_email (subject, sender, recipients, text_body, html_body):
A helper method for the send_password_reset_email. It specifies all information required to send an email.

def send_async_email (app, msg):
A helper method for the send_email function.

forms.py

This file contains classes of FlaskForm and specifies attributes of each form.

user.py

This file contains methods to update the user profile.

Functions

def get_reset_password_token (expires_in):
Return an encoded token to reset the password.

def verify_reset_password_token(token):
Verify that the user exists and reset is allowed.