ECE1779 Assignment 3
Community Blogs Platform

Xiaoxi Yu	Jiani Jia	Shuya Wang
1005654748	1002226245	1006549503

Submitted on December 10, 2020

1 Account Information

All aws services are implemented on the aws account: **april.yu@mail.utoronto.ca**

2 Introduction

Nowadays, as the rapidly spread of COVID-19, people tend to keep a physical distance between each other and stay at home. Thus, sharing their life online somehow becomes a trend all around the world these days. Since the online apps usage is exploding recently, we design a website for community members to post and read blogs, which brings neighbors together on these special days. In our project, our websites has the following features:

- Post blogs: We can share our life by posting blogs on this website. It is a platform for community members to record their important moment and share it with everyone. After registering and logging into the web page, you are free to post any blog, you can also update or delete your previous blogs.
- Read blogs: Users can regard this website as a more convenient bulletin board. Since we spend more time staying at home, sometimes we may miss or not receive the announcements timely. When reading posts, we can get better acknowledgement of our community changes.
- Filter blogs: For a community, if all blogs are gathered in the same page, it is too difficult to find the needed one. If this website, users can filter their interested blogs by clicking the hashtag, the post creator's name or the post's title.

3 Application Structure

This application consists of two lambda functions: one hosts the main web application (user-side), and the other includes a separate S3 storage management process that runs in the background.

3.1 Main Web

- The lambda hosting the main web application is deployed with Zappa, and it will be triggered by HTTP requests to the web application.
- Both user and post data are saved in DynamoDB. The application directly interacts with the DynamoDB to put_item, get_item, update_item, delete_item, query and scan if required.
- User profile pictures and post pictures are saved in S3 bucket.

3.2 DynamoDB

DynamoDB is used for storing user and post information, detailed structure of the user and post table are as follow:

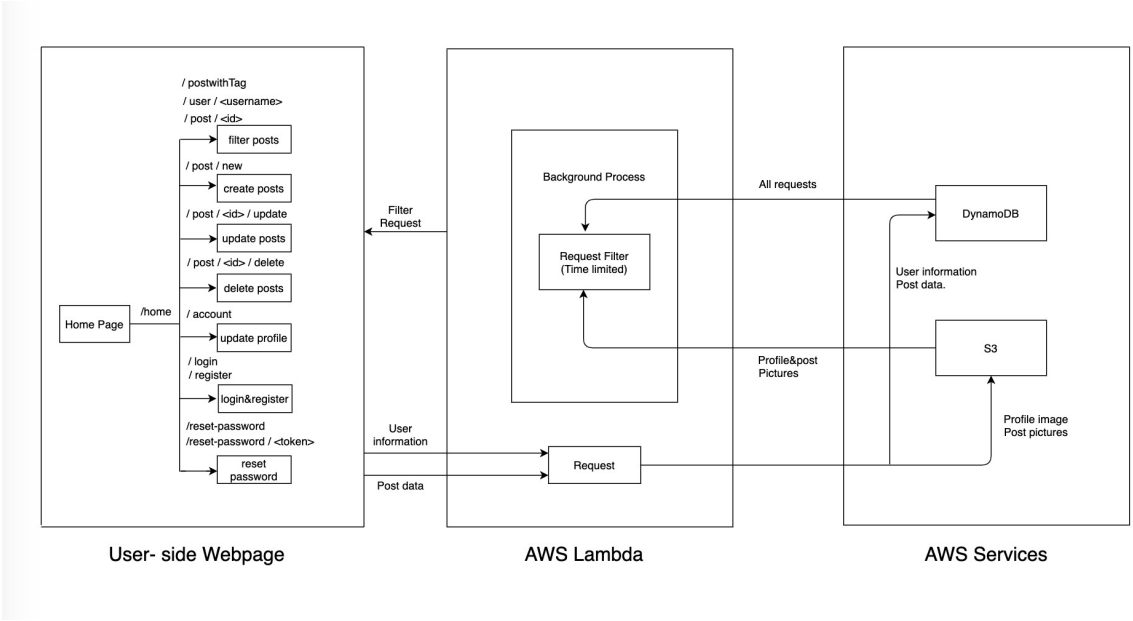

Figure 1: Flowchart for the Application Structure

3.2.1 User Table

- username: store the username for each account
- email: store the email for each account
- password: store the hashed password for each account
- image_url: store the filename for each account's profile image, and the profile image is stored in S3/s3bucket1997/static/profile_pics

3.2.2 Post Table

- id: store the unique id for each post
- title: store the title for each post
- content: store the content for each post
- tag: store the tag for each post
- image_url: store the filename for post image, and the post image is stored in S3/s3bucket1997/static/post_pics
- date_posted: store the datetime when post or update the corresponding post
- username: store the username of the creator for each post

3.3 S3

We create a S3 bucket **s3bucket1997** to store our user profile pictures and post pictures.

3.4 Background Lambda Process

The background lambda deploys a garbage collection. This function is invoked when our DynamoDB changes, and it will scan the database, deleting the post that does not meet the requirement. In our website, the post whose length is bigger than the max length will be deleted by the background process. We propose this idea since the we need to save the storage of our DynamoDB database, and in most social apps, the blogs with too long content are pointless and therefore forbidden.

4 User Instructions

We first create a zappa settings JSON file using the command of **zappa init**, then deploy the app into lambda function by using **zappa deploy dev** command. The JSON file will be used any time we try to update our application by using **zappa update dev** command. Once the webapp is deployed successfully onto the lambda function, it will generate a weblink in the terminal that will link to our web app directly. After entering the web page, then follow the instructions below to realize the full functionalities of this web app.

1. After clicking on the link, we will first enter the home page, users can see all the posts that have been posted on this web by other users, apart from the post contents, users may also see other users' icons, their posts' posted time and their posts' tags,however, if they want to post their own post, they need to login.
2. In order to login, you need to register an account first. Fill out all the form areas in this page, including your username, email, password and your profile photo. All the form areas are required fields and missing any part will lead to an unsuccessful register.
3. If a user forgets his password or wants to reset his password, he can go to the login page and click the blue link "Forgot Password?" and fill in the reset password form. Then an email with reset password link will be sent to his inbox.
4. After successfully registering, a message will flash out saying that "You have registered successfully." and the web page will route to the login page. You can access the home page by login to your account using email.
5. Once we login, we can post our own posts by clicking on the right-side button **'New Post'**. All the form areas are compulsory and needed to be filled in, including title, content, picture file and tag of this post. Submit the form and a message will be flashed as a reminder of success. Then we will be redirected to the home page, our newly created post is on the top of the home page.
6. For every posted blog, we can click the creator's username to check every blog he or she has sent.
7. If a user clicks the title of a blog, he will be led to a page with his clicked blog (only this blog). If he is the creator of the blog, two buttons will be displayed, one is "Update" and the other is "Delete". You can update the existing blog by refilling the post form, or you can just delete it.
8. On the right side, there are four tag lists, 'Life', 'Meme', 'Sport', 'Other'. Clicking on each tag, the web will take us to the page which has all the posts with the current tag. By doing this, we can help users to classify the information into four blocks so that we can filter out the information that the users are not interested in.
9. Also, click on the right-side button **'Account'**, all the account information will be shown at this page, and we can change the user profile photo by uploading a new local image.
10. Users can get to know information about our webapp through 'About Us'.
11. Users can log out by clicking on the log out button on the top right corner.

5 File Architecture

The primary web application allows users to register, log in, post blogs, update blogs, delete blogs, filter blogs, update user profile, retrieve user password through the email reset link. The files below will show its code structure.

1. run.py,__int__.py: These two modules are the initialization for our project.
2. create_table.py: This module creates two DynamoDB tables 'User', 'Post' for our project.
3. forms.py: This module defines the format and constraints of the LoginForm, Registration Form, UpdateAccountForm, PostForm, RequestResetForm, ResetPasswordForm. The constraints are listed as follow:
   (a) LoginForm: Email and password must be present to login.

(b) Registration Form: Username, email, profile photo and a pair of matching password input must be present to register. Number of characters of a valid username is between 2 to 20. Accepted formats of image are jpg, png, jpeg, JPG, PNG, JPEG. Duplicates of username and email entities are not allowed. All the inputs are compulsory.

(c) UpdateAccountForm: Profile photo file will be changed and is a required field. Username and email won't be changed but they are compulsory as well.

(d) PostForm: Title, content, picture, tag input must be present to post. Accepted formats of image are jpg, png, jpeg, JPG, PNG, JPEG.

(e) RequestResetForm: Email input is a required field.

(f) RestPasswordForm: A pair of new matching password inputs are required fields.

4. models.py: This module defines all the functions related to dynamoDB we need to use in routes.

5. routes.py

(a) home: Get all posts from the Dynamodb database and stay in home.html. Once the user is logged in, show all the posts and functional buttons. If the user is logged out, only show all the posts.

(b) postWithTag0(1/2/3): Query all the posts with the same tag out and then render templates to postWithTag.html.

(c) login: If a match is found for the email entered, the user will be allowed to login. If the password doesn't match the record or if the email doesn't exist inside the database, access will be denied with an error message shown on the page.

(d) register: If the username and email doesn't exist inside the database, they can be used to register. Upon successful registration, username and bcrypt hash of the corresponding password would be uploaded onto the database. Profile photos will be uploaded onto the s3 bucket and saved in the static/profile_pics folder.

(e) account: Users can upload image files to change profile photos, use update_item function to update the current user 'image_file' item in Dynamodb database.

(f) new_post: After getting all the data from the form in html, the app first saves the uploaded file into s3 bucket and then uses the put_item function to put current post information into DynamoDB database. Once the new post is created, the web will be redirected to the home page.

(g) post: Use get_item to get a post that has current id and stay in post.html.

(h) update_post: Get all the information of the update post from the PostForm, use the update_item function to replace the new contents of the post with the old ones in DynamoDB database.

(i) delete_post: Use delete_item function to delete current post.

(j) user_posts: Use username and get_item function to filter out all items with the same username attribute, render templates to user_post.html.

6 Cost Model

For the cost model, the resources we used are DynamoDB, S3 bucket and AWS Lambda.
We assume that one user makes at most 1000 http requests per day in the web application, the cost results are shown below.

Service Name	10 Users	1000 Users	1000,000 Users
Lambda	0.07 USD	7 USD	7000 USD
S3	0.0197 USD	1.97 USD	1970 USD
DynamoDB	0.15 USD	15 USD	15000 USD
Total cost	0.24 USD	23.97 USD	23970 USD

Figure 2: the cost after one month

Service Name	10 Users	1000 Users	1000,000 Users
Lambda	0.42 USD	42 USD	42000 USD
S3	0.1182 USD	11.82 USD	11820 USD
DynamoDB	0.9 USD	90 USD	90000 USD
Total cost	1.44 USD	143.82 USD	143820 USD

Figure 3: the cost after six months