COMS6998 CLOUD COMPUTING

FINAL PROJECT REPORT - SPRING 2022

Fashion Inspire – An Outfit Posting Platform

Front-end: https://github.com/Calypso52/photo-sharing-platform

Back-end: https://github.com/LegenQS/photo-share-platform

YouTube link: https://youtu.be/a2aHC4Edwkk

Yuting Zhou yz3917,
Xintong Liu xl3121,
Qishuo Wang qw2360,
Zhiqing Yang zy2491

Abstract

In this project we designed and built a fashion photo sharing platform and deployed this system on Amazon Web Services(AWS). The whole project was divided into three parts, front-end, back-end and database. The front-end is implemented based on react.js, providing platform users with separate account system which recommends outfits posted by other users according to their browsing and like history. The back-end was designed to process requests sent by the front-end. It applied algorithms to analyze the potential style that a particular user would like and fetch them from the database, responding to the web page. The database consists of three parts, namely s3 bucket, OpenSearch and DynamoDB. According to the functions they are good at, the three store photos, photo labels and post details respectively. Our goal of this system is to provide consumers with the perfect platform to share what they wear and recommend outfits based on user preferences.

http://photo-sharing-platform-frontend.s3-website-us-east-1.amazonaws.com

1. Introduction

"What should I wear today?" This is a question that many people ask themselves everyday. From personal style to trends to dress codes, it can be hard to know what to wear to school, work, and parties. It takes time to browse through e-commerce websites or go to the shopping mall to style up clothes. People usually browse through different applications to look for ideas for clothing styles, such as Instagram, Facebook, etc. On these social media platforms, users are modeling after influencers through tags, reposts and OOTDs (Outfit Of The Day), producing volumes of advertising content (**?**). However, existing social media such as Instagram have all kinds of contents, but not specifically on clothes, so it's hard to find the kind of style we want.

This project aims to solve the problem by more directly focusing on searching daily outfits, and aiming to facilitate the user experience by recommending with the styles they like. Fashion Inspire is a platform that allow users to share or find outfit ideas and inspirations. Some key features of the platform includes:

- When users log in, the home screen will show recommendations of clothing
- Users can search for photos using keywords such as "summer outfits"
- It recommends user relevant photo posts based on their previous behaviors (search, viewing history, likes, etc)
- Users can like a post, and it will show up in the "my favorites list"
- Users can post photos of clothes or outfits on the website, and add a description of the photo

2. Architecture

The architecture of the application is shown in figure 1. We will describe the details of database model and lambda functions below.

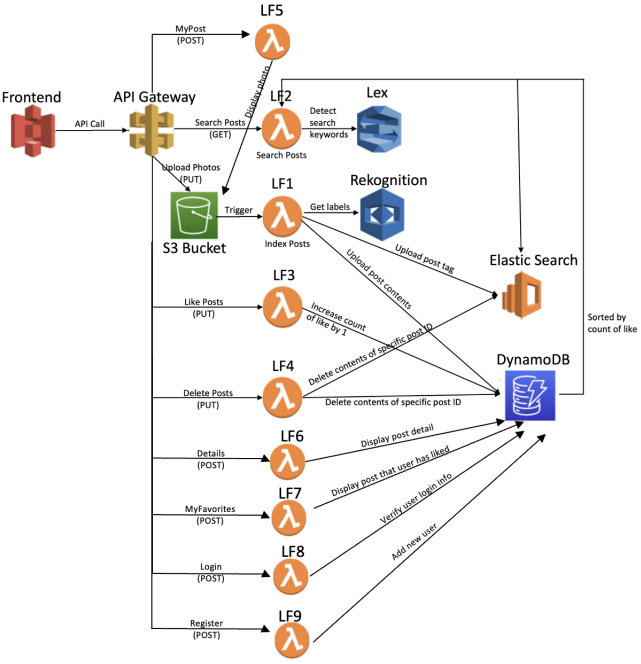

Figure 1. Architecture

2.1. Database

Our application uses three DynamoDB databases. *User* table stores users' username, password, their favorite posts and their own posts; *Post* table stores all the information of the posted post including its id, posted user, description, its title, labels as well as the 'user_id' that liked this post; *History* table stores all the viewing history of users to be used for recommendation. Since we utilize a combined method to make recommendation, we store posts that the user liked, posts that the user looked into details and the posts that user searched.

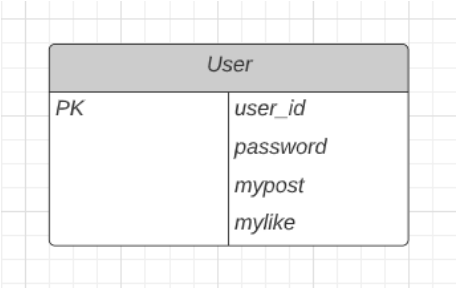

Figure 2. User Database Schema

Figure 2 shows the attributes in our *User* table. Username is set as the primary key that we can uniquely identify each user, and *mypost* has a list format which stores all the 'photo_id' that the user posted. We design this attribute for users to easily find their posted posts, so they can delete the post directly if they don't want it or they can see how much users has liked their post. Attribute *mypost* is used to store all the 'photo_id' that the user has liked, so they can look through all their liked posts by simply clicking *mypost* button in the front-end user main page, and they can also revoke their like on those posts.

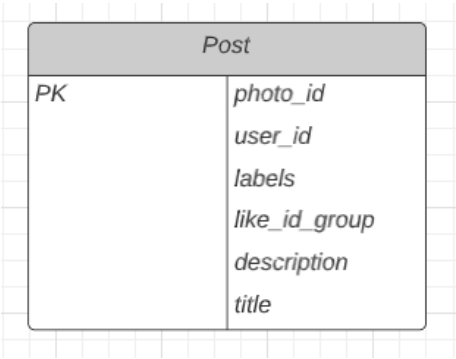

Figure 3. Post Database Schema

Figure 3 shows all the information stored by *Post* table. Here the 'photo_id' is identified by the object key in s3 bucket since it is unique, and we store the poster id, labels which are generated by amazon rekognition, like_id_group which is an empty list by default and stores all the users that liked this post, description that is attached to the post by the user and the title of the post.

Figure 4 shows all attributes of *History* table that our recommendation algorithm refer to. We have one record for each user, along with the posts they like, search and look into details. Besides the 'photo_id', we also store the labels of those posts to get the features to recommend. To better fulfill users interest, we focus on the recently updated records instead of the whole records.

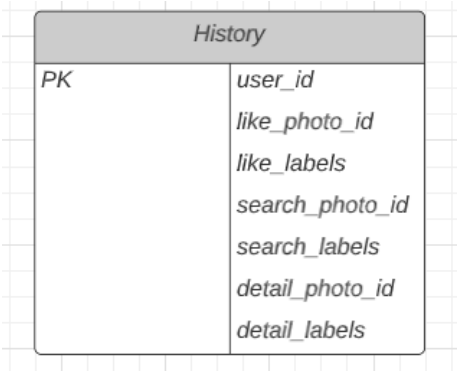

Figure 4. History Database Schema

2.2. Lambda Functions

2.2.1. Register

As an online platform, we provide users to register on our app to get full experience. Users can register with their username and password, then their information will be created in our 'user' and 'history' table for their login and our recommendation. Since we provide the password length and match check in the front-end and username check in the back-end, users will only be allowed to register by matched password at least with length of 8 and unique username (e.g. their email address).

2.2.2. Login

After registration, you are allowed to log in our app with your username and password. If you provide the wrong username that doesn't exist in our records, you will be notified with a wrong username; if you input the wrong password, a 'wrong password' message will be shown. Only when you provide the correct username with the matched password can you enter our app to experience further functionalities.

2.2.3. Post

The application provides the user with a function of posting photos they like to the interface and sharing with other users. And this function is implemented in the lambda function of the back-end. The lambda function of 'post' is connected to the S3 bucket which is used to store the images of posts. The S3 bucket will trigger the lambda function when there is a new photo load into the bucket. In addition, except for the image, the post of the user may also include the user account, the title, custom labels, and the description. All this information is passed into the S3 bucket in the format of metadata along with the image.

After receiving the information of the post, the S3 bucket triggers the lambda function and passes in the request data. Lambda function receives metadata of image in the S3

bucket which is all information of the post. Except for the custom labels in metadata, the application also uses AWS recognition to classify the image labels. The min confidence we set up for classification is 60 percents, and the recognition will return 10 labels based on the input of the photo.

All the information of the post will be recorded in Elastic Search and loaded into DynamoDB after the process of data collection and image classification. For the Elastic Search, photo ID and labels of images will be stored in it. Elastic Search is used to query particular images that can match the searching keyword of labels. If the searching keyword includes the labels of Elastic Search, then the relative ID of the photo will be returned back to the lambda function. This process of searching will be discussed more in the next section. For DynamoDB, all the relative data of the post will be insert into the post table which considers photo ID as the primary key. Meanwhile, the *mypost* attribute in the user table are also updated by inserting the new photo ID of the post into the list of previous posts.

Finally, the lambda function will return photo ID as the response back to the front-end. Then the front-end uses the photo ID to call the other interface related to this photo ID.

2.2.4. SEARCH

The 'search' function of the application is to look up particular photos based on the searching keyword entered into the searching box in the front-end. When the user clicks the search button in the frontend, the user account, and searching content will be passed into the lambda function as the request data. The back-end has created a lex bot to recognize keywords from searching content. For example, if we enter 'Show me Nike', the lex bot will recognize 'Nike' as the keyword and use it to search for photos from Elastic Search later. Also, we use the 'Inflect' package to transfer keywords in the form of plural into the singular form, so that the same keyword will not be searched for twice in two different formats.

Based on the classified keyword from searching content, the application uses it to query the matching labels from Elastic Search. And Elastic Search will return all photo IDs that corresponding labels can match the searching keyword.

The matching photo and its corresponding labels will be recorded in the table of history under the specific user account. All the matching photos and labels are appended to the list of attributes 'search photo id' and 'search labels'. The searching history of the user is recorded because the application provides recommendations for the user based on the history of preference. The idea of recommendation will be discussed more in the next section.

The lambda function will return the detail of posts back to the front-end based on all photo IDs it collects and their image URL. Then, the front end will redirect to the homepage interface to demonstrate the details of all these matching posts.

2.2.5. RECOMMENDATION

The function of recommendation in the application is used to provide suggested posts to users when they are direct to the homepage. The recommendation is based on the table of history under the particular user account. After receiving the user account from the front end, the lambda function looks up attributes of 'like labels', 'search labels', and 'detail labels' under this user ID. In other words, this function focus on the photos that users like, search, or view detail before. All labels of photos relative to the user's historical preference will be collected together. Then, we randomly choose 10 labels from them and search for matching photos from the Elastic Search.

However, if the user is in the first-time login and without any history of searching, liking, or viewing detail of posts, the application cannot search for any matching photos from the Elastic Search. In this case, the back-end will randomly pick some posts from the database and return back to the homepage in the front-end.

After collecting all the photo IDs of posts that return to the user, the lambda function format the return data and make it consists of the attribute of photo ID, image URL, amount of like, and the condition of whether the user has liked this post. All this information of posts will be returned along with the photo back to the front end. Then, the user can use the function of recommendation to like, search, or view detail of the post they prefer.

2.2.6. DETAIL

After logging in or refresh the main page, we will show the recommended posts for you, where there are the posted photos and you 'like' status. We allow users to click on the post to get more details. After clicking, you will be led to the main page of the post with its photo, title, description, post user and how many people liked it. Based on our recommend algorithm, once you click the information of the post it will be recorded in our 'history' table in the attribute of 'detail_photo_id' and 'detail_labels'. Then we will correspondingly recommend those posts with similar styles of the post you have looked through.

2.2.7. LIKE

In order to let us know more about your interests and keep your favorite records, we allow users to tick their favorite posts with a 'like' sign. After you 'liked' a post, it will appear in *mylike* where the user can see all your previously

liked posts in the user page. And if you don't like the post anymore, you can revoke your like operation at any time. In our database system, if you like a post, the 'photo_id' attribute will be stored in your *mylike* schema; if you don't like it anymore, the 'photo_id' of that post will be removed, which ensures that we only recommend relevant posts for you with the style of your liked posts. For a like request, we should judge whether the user has already liked the post or not, if so then the current request should be 'unlike' where we should delete information from databases, or it would be a like request where we will add information. Specific procedures are displayed by figure 5.

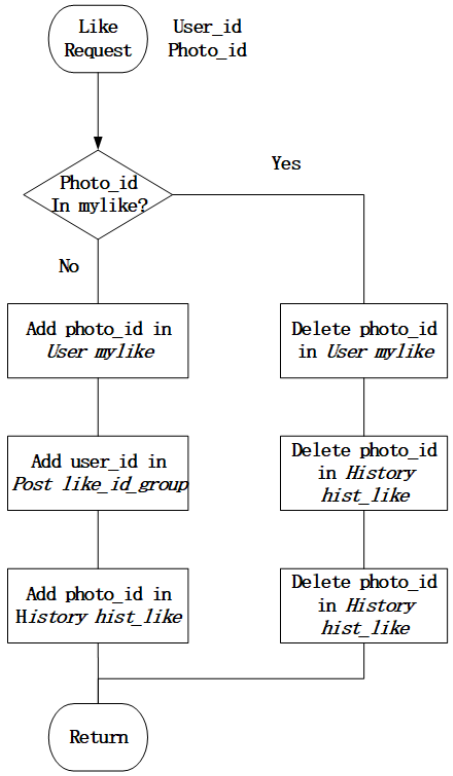

Figure 5. Procedure of Like Request

2.2.8. DELETE

To be more user friendly, we enable users to delete their own posts. Firstly we check if there is any user liking the post. If not, we can simply delete the post from our 'post' database directly. If so, we will get all the users that liked this post, and delete the post from their *mylike*. After that, we will also delete the record from 'user' database in the attribute *mypost*. Finally, the photo will be deleted from s3 bucket. You can see the detailed procedures in figure 6 below.

2.2.9. MYPOST

After posting a photo on the platform, you post information will be stored in our database. You can go directly to your

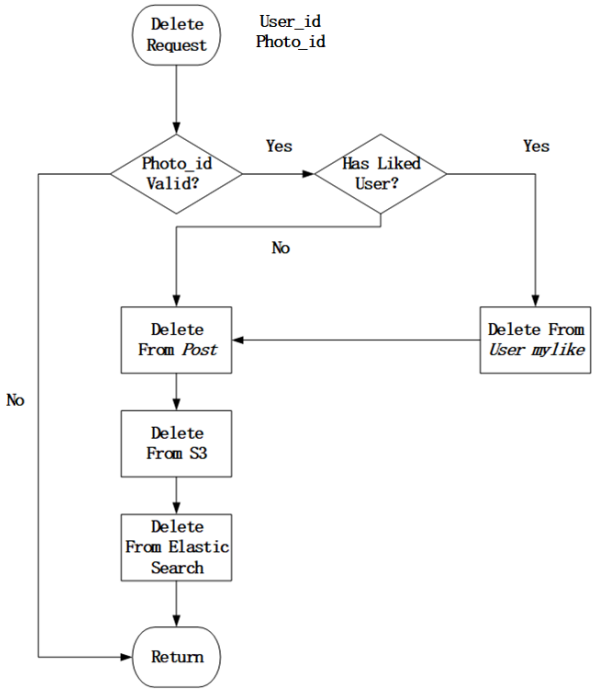

Figure 6. Procedure of Post Delete

user setting page, and all your posts will be displayed when clicking 'mypost'.

2.2.10. MYLIKE

Similar to mypost, with the functionality of lambda function 'like', all your liked posts will be stored and you can also go to your user setting page to get all posts you like by clicking 'mylike'.

3. API Designs

Here are the 10 APIs we implemented on AWS Api Gateway.

3.1. POST/signup & POST/login

The login and signup interface is used to verify whether the account information is correct, and if correct, navigate the user to the platform main page. They take the account and the password as the request data from front-end to back-end, responding with whether the operation successes.

3.2. PUT/post

In the publishing posts interface. When the user edits a post by uploading a picture, writing title, labels and text information, and clicks the post button, the post interface is called. The picture will be stored in the s3 bucket, and other text information will be recorded in the form of metadata in

s3. The lambda function gets this information by setting the trigger to the s3 bucket.

3.3. POST/search

The most important thing is the 'search' interface. All pages have a search box at the top. When the user enters text content in the search box and clicks the search button, the search interface is called. Then search results will be returned and presented on the main page.

3.4. POST/recommendation

Once the user enters the main page, the 'recommendation' interface will be called. The interface takes the account name as the request body, and the back-end returns the recommended posts based on the user's search and like history. we engage users with such smart recommendations. There is a button at the top of each page to go back to the main page, which represents getting recommendations through this interface.

3.5. POST/detail

Users can freely click on the photos on the page to view post details. At this point the 'detail' interface will be called.

3.6. PUT/like

On the picture details page, users can like a picture by clicking the heart shape icon. This operation is recorded into the database through an interface named like, and is also recorded as one of the factors for recommending pictures.

3.7. POST/delete

The last interface is the 'delete' interface. On the user's personal information page, all the posts uploaded by the user are displayed. There is a delete option on each post, after clicking, the 'delete' interface is called.

3.8. POST/myposts & POST/myfavorites

In the user account interface, there are two operation options, one is to display the posts posted by the user by calling the *myposts* interface, and the other is to display all the posts that the user has liked by calling the*myfavorites* interface.

4. Project Detail

In this section, we will show how a user can interact with the platform. First, user can sign up and log in in the login page (figure 7). Once logged in, the user will see a page of recommended photos. If this is the user's first time logging in, random photos will be shown (figure 8). User can like a post by clicking the heart button once hovering over the post. When user logs in the next time, the application will remember the liked content and the front page will show similar items (figure 9). User can also directly search for contents using the search bar.

If the user wants to post some photos about outfits and personal style, he/she can click on the icon on the bottom right side and be directed to the post page (figure: 10). User can upload a photo, and optionally write title, customized tags, and description for the post. Posted content will be shown on the Mypost page.

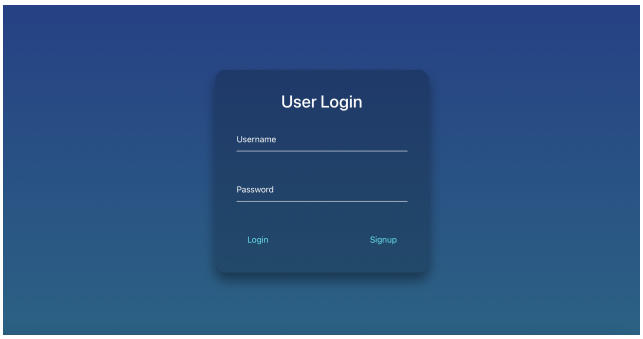

Figure 7. login page

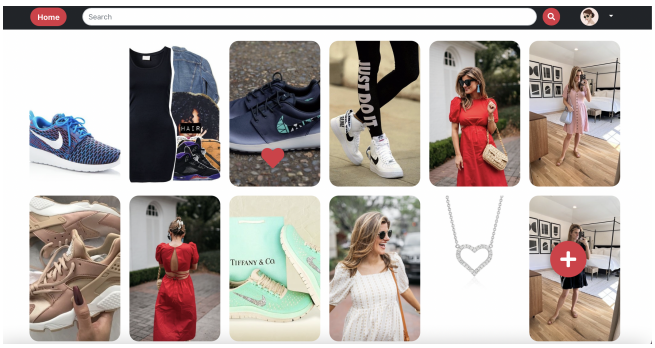

Figure 8. front page

5. Conclusion

We developed a fashion photo sharing platform on AWS for people to post and get recommendations based on their browsing and liking histories. Our front-end framework is an extensible framework, which means that when adding functions according to user needs in the future, developer can select the corresponding template for rapid expansion. Our back-end is implemented using AWS lambda functions. We used algorithms described in chapter 2.2 to extract labels of pictures posted by users, fetch the recommendation posts based on user history to attract them. There are still some potential improvements for our project. The logic

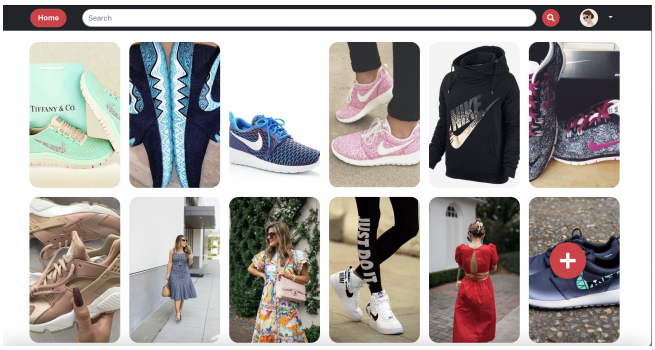

Figure 9. front page after liking a picture about shoes

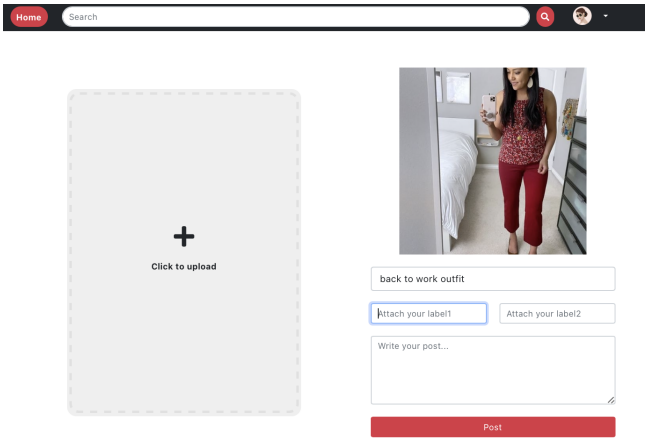

Figure 10. Post can be created here

of recommendation in our system is that we compare the labels of the photo contained in the user's browse and like history with the photo containing the specific label in the database, and return if there is a match. As a matter of fact, the recognition result may contain over thirty related labels. The algorithm should have the ability to distinguish the most relevant labels that the user would like to be recommended instead of returning all photos if one of the labels are hit. Another improvement is to expand the database. Currently, we collected hundreds of pictures in the database. The bottleneck of expanding the database lies in two aspects. One is that it is not easy to collect enough images from the internet with nearly every labels are covered, to confirm that no matter what style of picture that a user would like, the system will always has the corresponding ones storing in the database. The other one is that we should discover a method to upload such a huge amount of images from the aspect of the user. That's to say, only by sending posts from the front-end can we enrich and classify different pictures. We must figure out a way to import pictures in batches from the perspectives of different users to complete the original accumulation of the system database.

6. Contribution

Yuting Zhou (yz3917): Frontend, paper
Xintong Liu (xl3121): Backend, paper
Qishuo Wang (qw2360): Backend, paper
Zhiqing Yang (zy2491): Backend, paper

References

[1] Abidin, C. Visibility labour: Engaging with Influencers' fashion brands and #OOTD advertorial campaigns on In-stagram. Media International Australia, 161(1):86–100, 2016. doi: 10.1177/1329878X16665177. URL: https://journals.sagepub.com/doi/10.1177/1329878X16665177

[2] React document. https://reactjs.org/

[3] Deploy ReactJS App On AWS S3. https://blog.cloudthat.com/step-by-step-guide-to-deploy-reactjs-app-on-aws-s3/

[4] React Router. https://reactrouter.com/