COMS 6998 - Big Data and Cloud Computing
Columbia Exchange Platform

Yuqi Zhang (yz3983), Weiran Wang (ww2584), Dongbing Han (dh3071), Jiapeng Guo(jg4403)

Abstract

"Columbia Exchange" is a platform that supports Columbia students to sell the stuff they don't need to other Columbia students. The target users of this application would mainly be the Columbia students who want to buy cheaper products or sell their second-hand stuff. "Columbia Exchange" is being restricted for Columbia students, therefore, only the students with active Columbia email accounts can register with it. This application is being developed using AWS Cloud Services, like Cognito, RDS, etc.

Introduction

Every year, there are many Columbia students graduating and there will be a problem with their second-hand furniture/stuff. The purpose of this application is to effectively help Columbia students to sell the stuff they do not want or buy cheaper products from a reliable source.

Architecture

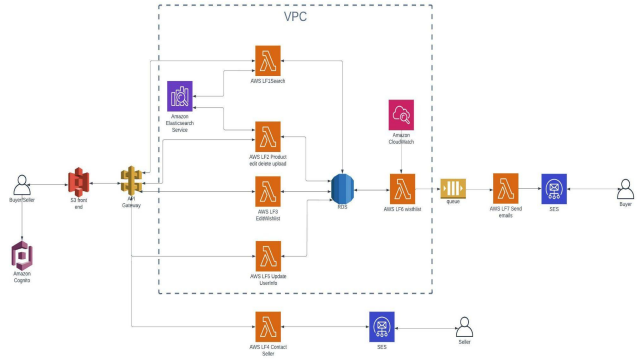

Fig. 1 Project Architecture

We used Cognito to authorize users and S3 to host the frontend. We also have an API Gateway to connect the frontend and backend. We used a RDS Database to store our data and put all the lambda functions that need to query data in the same VPC with RDS Database. We also used some other services like SES and SQS to send email to users.

API Design

Our API Gateway has 7 endpoints, **/productsearch** (GET), **/upload** (POST, PUT), **/upload-img-s3**(PUT), **/wishlist** (POST, DELETE), **/contactseller** (POST), **/getuseritem** (POST), **userupload**(Post).

1. For the **/productsearch** endpoint, the user would send a GET request with a query "q" as a URL query string parameter through API Gateway to the LF1(product-search) which would use the query to fetch through the OpenSearch to find the products with the same tag. And LF1 will also use that query to fetch through the RDS database to get products with similar names. This detailed information of those matching products will be returned to and displayed on the frontend.

2. There are two methods (POST and PUT) for **/upload** endpoint. Both of them are connected with the LF2 (Product upload/edit/delete). When the user wants to upload a new product, the frontend will send a POST request and LF2 will update all the related tables. Similarly, if the user wants to edit the product, the frontend will send a PUT request to update all the related tables. Deleting a product in our application will not cause a real deletion, but just update the "is_deleted" column in the product table to mark that product as deleted. The LF2 would also index the product in the elastic search if the user uploads a new product.

a. LF1 would first check if the user wants to create a new product or edit an existing product. To create a new product, first add the product information to the RDS, and index the product in elastic search.
b. To edit the product, we should use the product_id to to delete the product information from the RDS and elasticsearch.

3. The **/upload-img-s3** endpoint is always called at the same time once a user successfully triggers LF2 and upload information of a product to RDS. It will assign the product a pid using the **uuid** library in Python, and rename the image file with the pid to connect that image with the product. Then it will upload the corresponding image to a s3 bucket and the database saves its link of the corresponding object name.

4. The **/wishlist** endpoint has three methods: GET, POST and DELETE. Both of them are connected with LF3 (Editwishtlist) which would get, create or delete the user's subscription of a tag. When sending the GET request, it takes in a query parameter, uid, to find all the tags that user subscribed to. When sending the POST or DELETE request, the user_wishlist table in the RDS database will be updated.

5. /contactseller endpoint will send a POST request with the message they want to send (username, email, phone number, message as well as seller's

information) to the API Gateway, then the API Gateway would connect with LF4, form an email and send the email to the seller using SES.

6. /userupload endpoint will connect with LF5 (UpdateUserInfo) and update user's information once a user successfully registered. This endpoint will send a POST request with the user profile information from Cognito (username, email, description) to the API Gateway, then the API Gateway would connect with LF5 and store the information into the RDS database.

7. The last function does not have an endpoint trigger but a CloudWatch trigger. We set a weekly CloudWatch trigger that would trigger LF5 to get all tags in the wishlist of the current user. After getting all these tags, LF5 will send these tags to AWS OpenSearch, then it will return the related productIDs. After getting these pids, we will connect to RDS in order to get the detailed information of those matching products. Then we will form an email and push it as a message to SQS and wait to be sent out. There is a LF7 (wishlistEmails) which is triggered by an SQS event. When the SQS receives a new message, LF7 will poll it and send the email to the user using SES.

Some API descriptions

/productsearch (GET)

— Request Body (Parameter)

```
{
    q : string
}
```

— Response Body

```
{
    statusCode: "200",
    body: {
        "product": productDetail,
        "user": userDetail,
        "tag": tag
    }
}
```

/upload (POST, PUT)

/POST

– Request Body

```
{
    "name": string
    "description": string
    "price": int
    "tag": string
    "Image": string (url)
    "seller_email": str
}
```

All fields are required

– Response Body

```
{
    statusCode: "200",
    body: {
        product_id: string
    }
}
```

Item_id is required but will be automatically generated. Other fields are optional depending on if the user

wants to change what attributes. Status can be changed to '1' if the user decides to delete the listed item. And thus there will be a change in lambda function.

/contactSeller (POST)
- Request Body
{
    "seller name": string,
    "seller email": string,
    "user name:: string,
    "user email": string,
    "user phone number": string,
    "message": string
}
-Response Body
{
    statusCode: "200"
}

/wishlist (POST, DELETE)
/POST
- Request Body
{
    "uid": string,
    "tag": string
}
-Response Body
{
    statusCode: "200"
}

/GET
- Request Body
{
    "uid": string
}
- Response Body
{
    "statusCode": 200,
    "body": list of tags
}

Project Detail

Functionality Introduction

Firstly, our application restricts users to register with Lion emails, and once the user logs in, they are able to add or delete products on their selling list through our application and also view his/her selling list and other information in the user's profile. Secondly, our application also supports the user to give the product a tag when they upload it to their selling list so that others could find the product more easily. Also, for those registered users, they can subscribe to any tags that they are interested in and they will receive weekly emails of the newly added products about that tag. Moreover, our application supports leaving messages to sellers from buyers. Users can contact sellers on the application and ask questions about items they want to buy. Sellers will automatically receive information about who is interested in his products and which product has a potential buyer. Besides

those functionalities we described above, our application also supports users to search what they want by typing the keywords in a search textbox no matter if the user has registered or not. Users can also click an item to view the details of such a product.

Fronted Introduction

We have Index.html, confirm.html, signin.html and signup.html consists of the whole register, login and logout process which connect with aws cognito and RDS database to keep user profile information. Profile.html is used for a user to view his/hers information after login. And also support users to upload an item or check items that they already upload. Users can also select his/hers interested tag in this page and our application will automatically send recommended products regularly. Upload.html and edit.html supports users to upload a new item to our exchange application and edit/delete any of the items that they have already uploaded. The front page connects with the RDS database and our application will receive change of the item once the user successfully uploads/edits. Homepage.html is the main page of our application, we automatically randomly recommend some of the items to our user and allow them to view any of the items on that page. Users can also search by tag or name to view items. ProductDetail.html provides more detailed information after the user clicks any item in homepage.html. Information includes product's name/price/description/tags, and sellers information such as their username and email address. We also automatically recommend other items uploaded by the same seller on this page and only allow users to leave a message to the seller when they login.

Database Introduction

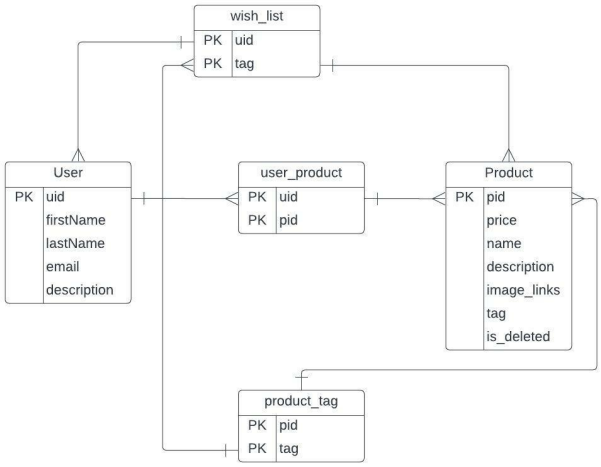

Fig. 2 Database Design

Fig.2 is the design of our database. We have 5 tables in a schema that is stored in a RDS database. The "**User**" table is gained from the Cognito User Pool. Everytime a new user registers, the user's information will be stored in this table. Once a registered user

uploaded a product, the product information would be stored into the "**Product**" table. "**user_product**" table and "**product_tag**" table are two relational tables that would also be updated. When a user adds any tag to their wishlist, the "**wish_list**" table will be updated. Following are some SQL scripts to create the tables in our database.

```
CREATE TABLE
`6998_final_proj`.`User` (
  `uid` VARCHAR(45) NOT NULL,
   `email` VARCHAR(128) NOT NULL,
  `description` VARCHAR(128) NULL,
  PRIMARY KEY (`uid`),
    UNIQUE INDEX `uid_UNIQUE` (`uid`
ASC) VISIBLE,
  UNIQUE INDEX `email_UNIQUE` (`email`
ASC) VISIBLE);

CREATE TABLE
`6998_final_proj`.`user_product` (
  `uid` VARCHAR(45) NOT NULL,
   `pid` VARCHAR(128) NOT NULL,
    PRIMARY KEY (`uid`));

CREATE TABLE
 `6998_final_proj`.`Product` (
  `pid` VARCHAR(45) NOT NULL,
   `price` VARCHAR(128) NOT NULL,
   `name` VARCHAR(128) NOT NULL,
   `description` VARCHAR(128) NOT NULL,
   `image_link` VARCHAR(128) NOT NULL,
        `created_time` VARCHAR(128) NOT
NULL,
    `is_deleted` NUMBER NOT NULL,
     PRIMARY KEY (`uid`));

CREATE TABLE
`6998_final_proj`.`wish_list` (
  `uid` VARCHAR(45) NOT NULL,
  `tag` VARCHAR(128) NOT NULL,
   PRIMARY KEY (`uid`));
```

Conclusion

In this project, we took advantage of numerous AWS services included but not limited to AWS Cognito for user identification, AWS S3 for hosting the website and storing pictures, AWS Lambda for connecting services, AWS RDS for storing user information and product data, AWS APIGateway for creating and managing APIs, AWS SES for sending emails to users, AWS Opensearch to support searching on the website, and AWS Cloudwatch to send weekly update to users. Columbia Exchange allows those affiliated to Columbia University to sell and buy used products. It also sends weekly digests to users who subscribe categories of items of their interest. Therefore, our website could potentially help Columbia students and faculties to exchange their used items, lower cost of living, prompt more sustainable lifestyle, and expand social networks.

YOUTUBE live demo:

https://www.youtube.com/watch?v=Ablhb5Jw7BE