# CS591E1 Final Report

Team Awesome: Qian Xiang, Tong Li, Yueying Li, Haiqi Ma, Kaiyuan Fan

## 1. Overview

Treasure Hunter is an application for users who want to find something they are interested in but don't know where to buy it. By using this app, users can simply take a photo and the app will let the users know details about the item including the price, the brand, the description and a link of the shopping website. If users are not satisfied with the results we provided, they can modify the feature tags we recognized by the Google Cloud Vision API and search again. If they are still not satisfied with the result, they can share the item photo to other media like Instagram and ask others for help.

Treasure Hunter has several functions, including a built-in camera for taking a photo or retrieving a picture from the gallery, cropping feature to highlight the interests in the image, image recognition, product searching and user's profile management.

## 2. App's Functionalities and Implementations

### 2.1 Login/out System and Profile Editing

Before starting a "hunting" journey, our app requires users to have an account (Figure 2.1.1). We support three login methods. Users can create an account (Figure 2.1.2) or use Google or Instagram to log in. After signing in, users can experience the whole function that we provide in this app.

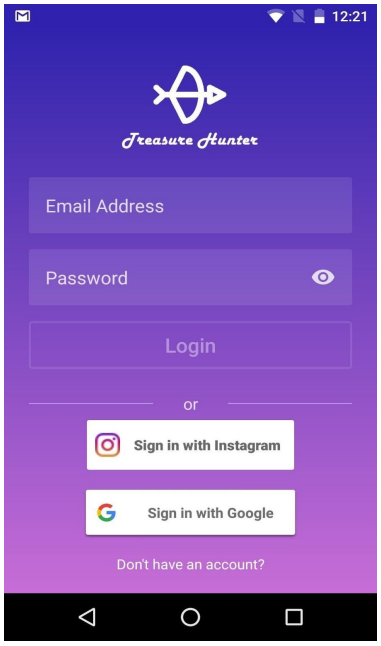



Figure 2.1.1 Login screenshot

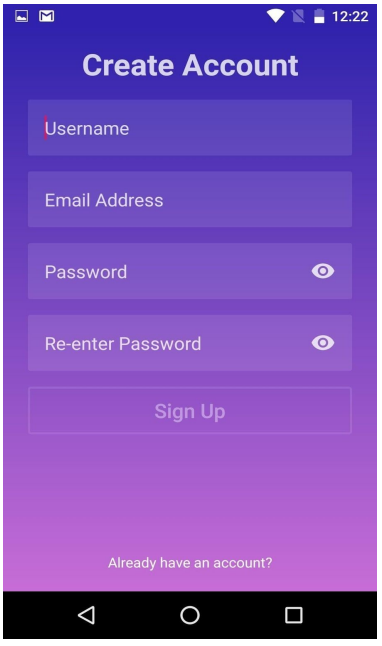



Figure 2.1.2 Sign up screenshot

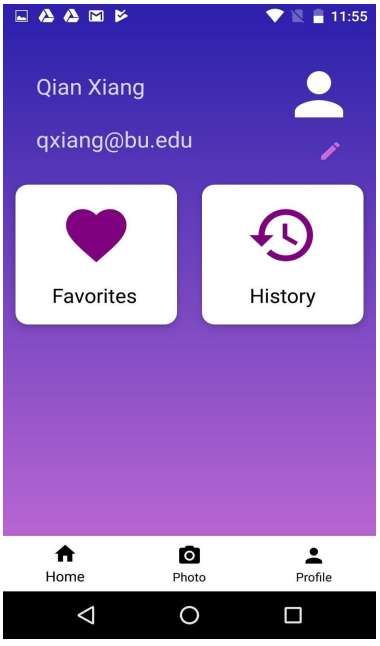



Figure 2.1.3 User's profile screenshot

In the users' profile page (Figure 2.1.3), they can review their basic information here. They can change information by clicking the "pen" below the profile image, and then entering the

profile modification page (Figure 2.1.4). In this page, Users can change their password, username, email address, and profile image. By clicking the confirm button, all the new information will be updated in the database. Users can also log out by clicking the "Logout" button and will be navigated back to the "Login" page.
Users cannot take photos or upload a picture until they log in but they can browse other users' posts on the home page.

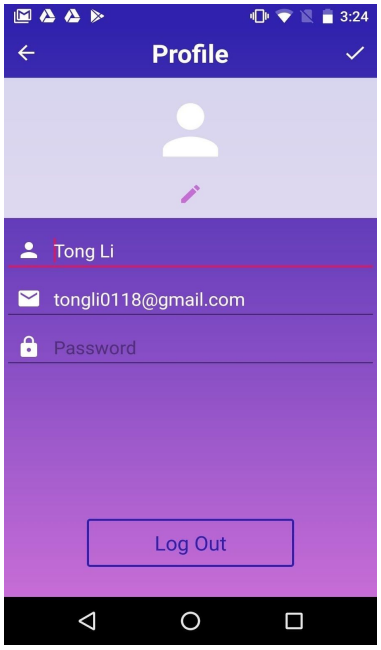


Figure 2.1.4 Modify user's profile info screenshot

Figure 2.1.5 History display screenshot

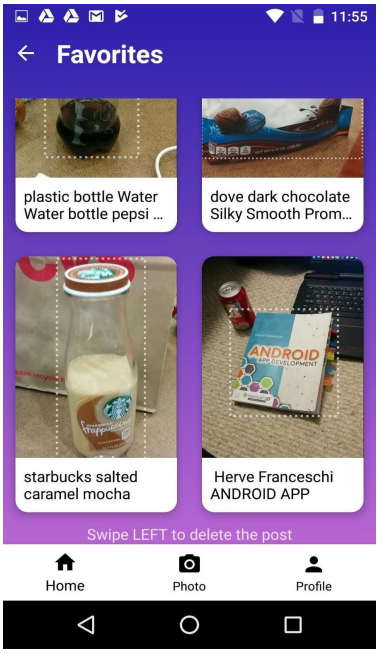
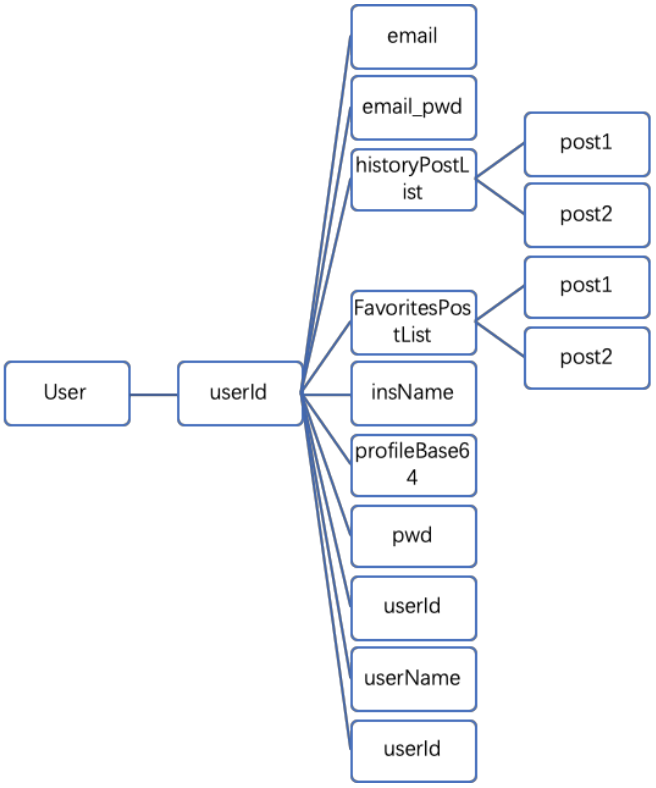


Figure 2.1.6 Favorite display screenshot

Back to the profile page, here are two sections below users' information, one is "Favorites" and the other is "History". The History folder (Figure 2.1.5) stores all the posts the user has posted before. And the Favorites folder (Figure 2.1.6) contains all the posts the user has liked. Users can simply remove the posts in both folders by swiping the post to the left.
For this user section, we use Firebase to store all the user's information. The structure of the user class shown below:




1) we create a user class for each account in our app
2) We add an item named email_pwd in order to match the email and password for the general user login which because of firebase doesn't support query nested search.
3) In order to search the user information, we add an onDataChange listener to grasp the user information which matches the condition. The basic searching code shows below:

```
databaseUser.orderByChild("email_pwd").equalTo(search)
.addListenerForSingleValueEvent(new ValueEventListener() {
@Override
 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
//if this user exist
if(dataSnapshot.exists()){
        cur_email = useremail;
        //jump to MeActivity
        for (DataSnapshot child: dataSnapshot.getChildren()) {
        globalClass.setUserId(child.child("userId").getValue(String.class));
        globalClass.setUserName(child.child("userName").getValue(String.class));
        globalClass.setProfileBase64(child.child("profileBase64").getValue(String.class));
         }
        globalClass.setLogInType(0);
        globalClass.setLogIn(true);
        globalClass.setEmail(cur_email);
        Intent i = new Intent(getApplicationContext(), MeActivity.class );
        startActivity(i);
}else {
Toast.makeText(getApplicationContext(), "User doesn't exist or wrong password.",
Toast.LENGTH_SHORT).show();}
 }
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
        }
 });
```

4) We use global variables to store the current user information which can be used in every activity through the whole app.

## 2.2 Camera

The camera has these basic functions: taking a photo, retrieving a photo from the galley, autofocus when users touch the screen and flash.

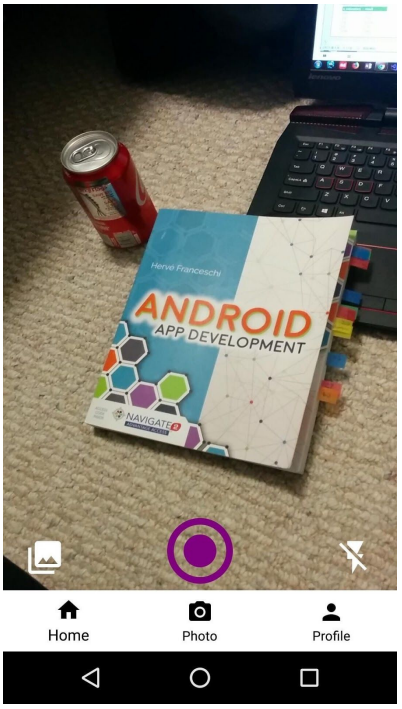



Figure 2.1 Camera screenshot

To build this camera, we use android.hardware.Camera API.

### 2.2.1 Taking photos

In order to take pictures with this class, we follow the instructions of the Android developers' document:

1) Obtain an instance of Camera from *open(int)*.
2) Get existing(default) settings with *getParameters()*.
3) If necessary, modify the returned Camera.Parameters object and call *setParameters()*
4) Call *setDisplayOrientation()* to ensure correct orientation of preview.
5) Pass a fully initialized SurfaceHolder to *setPreviewDisplay()*.
6) Call *startPreview()* to start updating the preview surface.
7) Call *takePicture()* to capture a photo
8) After taking a picture, call *stopPreview()* to stop updating the preview surface.
9) Call *release()* to release the camera for use by other applications.

### 2.2.2 Retrieving photos

There are two steps to retrieve a photo from the gallery.

1) Using *startActivityForResult()* to start an activity also set Intent as ***ACTION_GET_CONTENT.***

2) Call *onActivityResult()* to receive the result(the retrieved image).

2.2.3 Autofocus

There are three steps to add autofocus function in the camera.

1) Declare the app uses the android.hardware.camera.autofocus feature, in the *<uses-feature>* manifest element.
2) Get existing(default) settings with *getParameters()*.
3) Call *setFocusMode()* to set the focus mode as ***FOCUS_MODE_AUTO***

2.2.4 Flash

There are three steps to add flash function in the camera.

1) Declare the permission **android.permission.FLASHLIGHT**
2) Get existing(default) settings with *getParameters()*.
3) Call *setFlashMode()* to either open the flash by ***FLASH_MODE_ON*** or close the flash ***FLASH_MODE_OFF***

**2.3 Image Cropping**

In order to utilize the function, we mainly use two Drawables to represent the image to be cropped and the cropping rectangle box which users can set the size and move.

To set size, we override *onTouchEvent()* to update four corners' positions.

To crop an image, it's basically a process to draw a new image on canvas according to the four corners' positions.

After the cropping process, we pass two images to the image recognition activity. One is the cropped image only with the item that the user is interested in, and it will be used to recognize features and generate some tags related with the item, One is a larger image with a white rectangle box surrounding the item which will be used for later posting part.

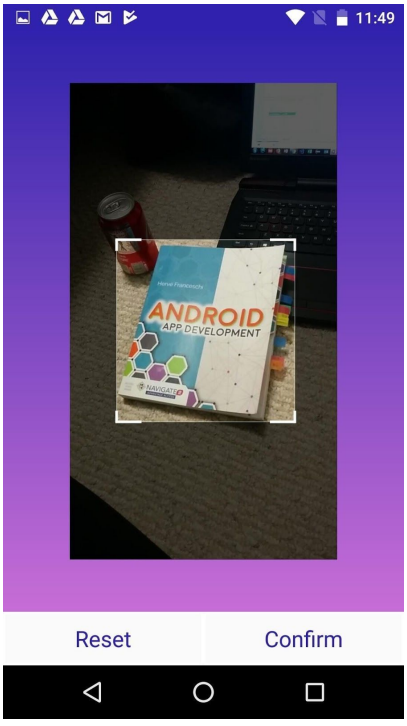



Figure 2.2 Image cropping page screenshot

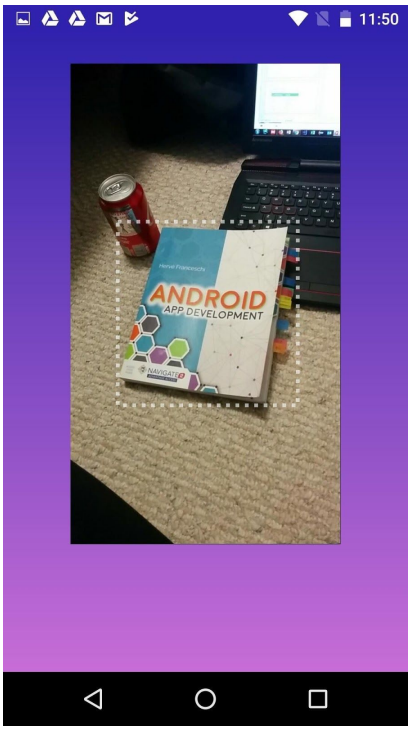



Figure 2.3 Image after copping screenshot

### 2.4 Image Recognition

After receiving the cropped image from the image cropping process, we use Google Cloud Vision API to get tags based on the content of it. There are three steps we interact with the Cloud Vision API:

1) We configure the Google API client (Vision class): specifying the API key, the HTTP transport (NetHttpTransport class, responsible for communicating with Google's servers), and the JSON factory (AndroidJsonFactory class, responsible for converting the JSON-based results the API generates into Java objects).
2) We specify the feature(e.g. Label, Web, Logo, and Text) we are interested in while making a request to Google Cloud Vision API.
3) Once the request has been processed, we get a BatchAnnotateImagesResponse object containing the response of the API and obtain tags form it.

For all of these tags obtained, we do tag filtering:  remove duplicate tags, remove null values and select tags with the longest match principle. Also, considering the reliability and simplicity of the tags, we only use features with a confidence score higher than 0.9 for label detection and ignore too short and too long features for text detection. we combine the selected tags together, passing them as the input keywords for further searching section.

### 2.5 Product Search and Selection

2.5.1 Product search

Product searching in our app is using the tags from the image recognition piece. After getting the feature keywords of the item, we use the external Serp (Search Engine Results Page) API to fetch the possible shopping item list. All the results are from the Google shopping searching website. Serp API is integrated by sending a formatted URL to request response of the structured JSON results.

Accuracy of intended item searching is mostly based on the feature keywords we get from the Google Cloud Vision API, enough though the API is powerful, it’s not perfect. We have implemented the feature for the user to modify the feature tags and re-fetch the shopping item list.

After we getting the response JSON file, we have to parse it into a clean item list with the item price, source, description, preview image, and shopping website. We displayed the item list inside a scrollable listview and users can click load more button to increase the item options. (Figure 2.51)

2.5.2 Best matching item selection

Upon all the shopping options in the list, users can pick the one they think it’s the best matching of the item in the photo. (Figure 2.52)

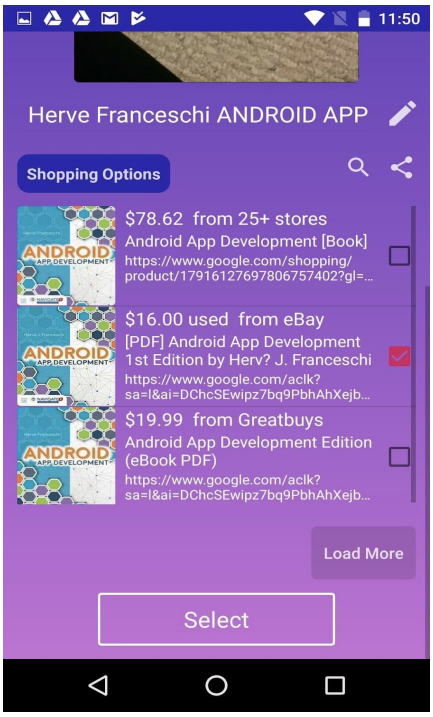



Figure 2.51 Select best match screenshot

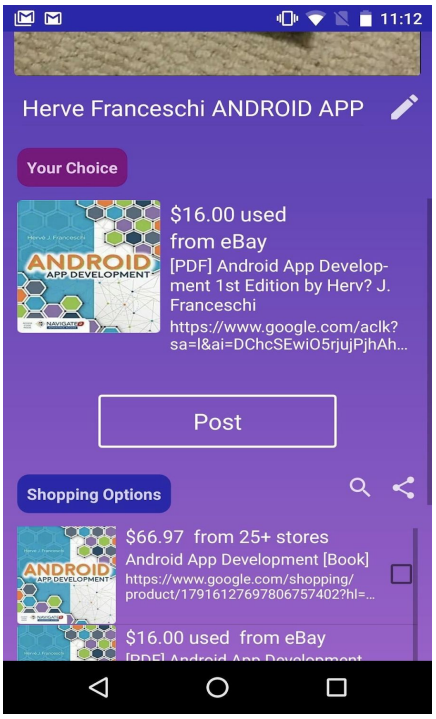
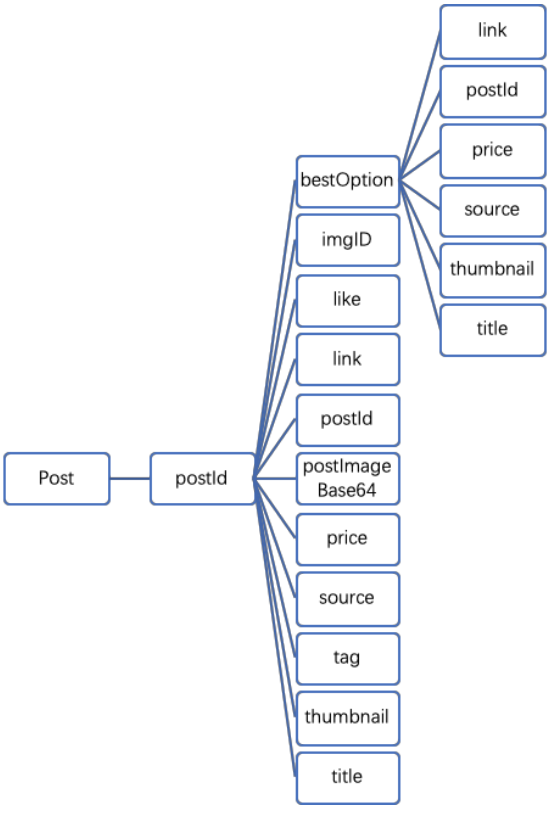



Figure 2.52 Best match selected screenshot

## 2.6 Post Generating

Once the user makes the best option selection, they can choose to generate the post and share it with the other users. This process will automatically add the post into their history category. Users can always access and manage their posts under the history category. The database structure of a post shows below:



## 2.7 Post External Sharing

If users are not satisfied with what our app finds for them, they can always post the picture to Instagram and ask their friends for help.

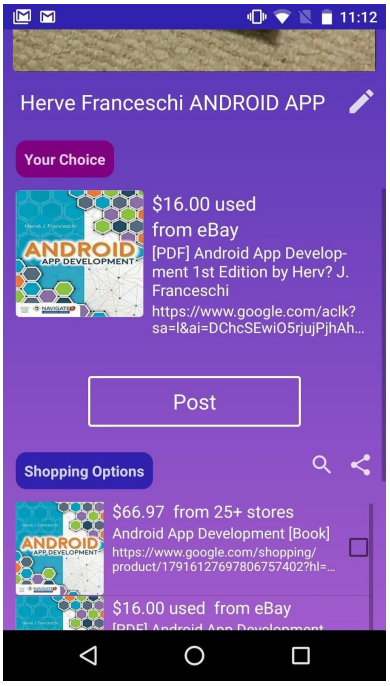



Figure 2.7.1 Before share screenshot

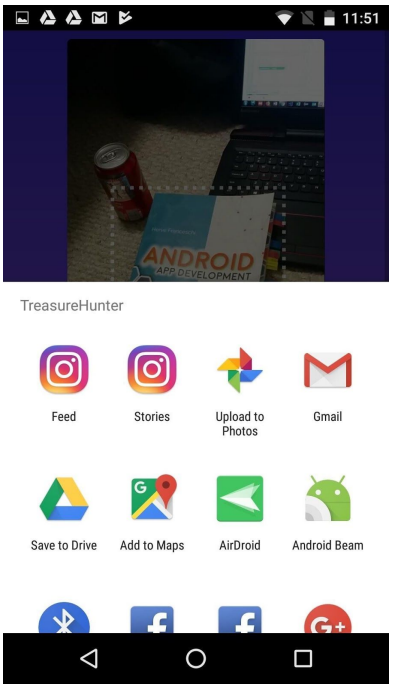



Figure 2.7.2 Share to Ins screenshot

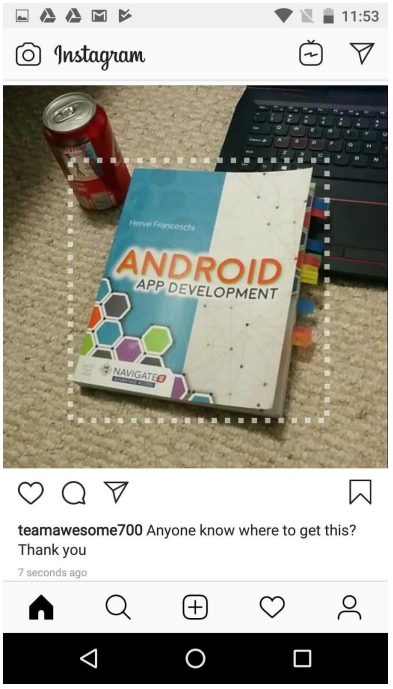



Figure 2.7.3 After share screenshot

Android apps are allowed to push video and photos to the Instagram app via Android Intents.

1) Create a direct share menu and add a share icon.
2) Inflate menu resource file and create the new Intent using the 'Send' action.
3) Add the photo URI to the Intent and then broadcast the Intent.

It creates an Implicit Intent which will then prompt the user to select the application they wish to send the media to. After pressing the share icon in the upper right corner, the share menu appears and users can share the picture either to the post or to the Instagram story. When triggered, Instagram will immediately present the user with the crop screen. If users haven't logged into the Instagram account before sharing the picture, the system would send a toast showing that you must log in before you can share on Instagram.

## 2.8 Home

The home page is mainly a page displaying what users have found via our app and it is the main page users would see if they open our app for the first time. In this page, we use RecycleView to present the results retrieving from the database. If users haven't signed in, the only function they could use is browsing other posts and clicking the post to see the details. If users signed in, they can search on this page and like or unlike other posts.

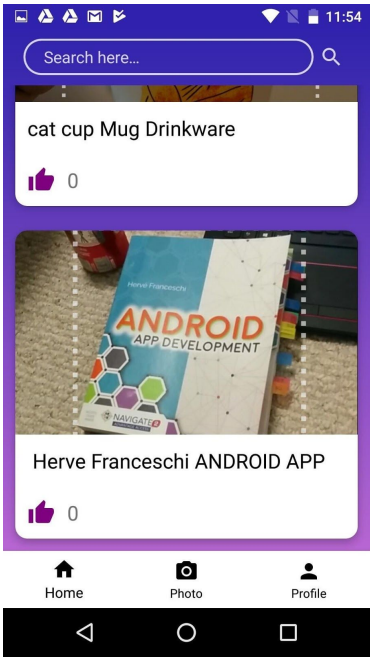



Figure 2.8.1 Home screenshot

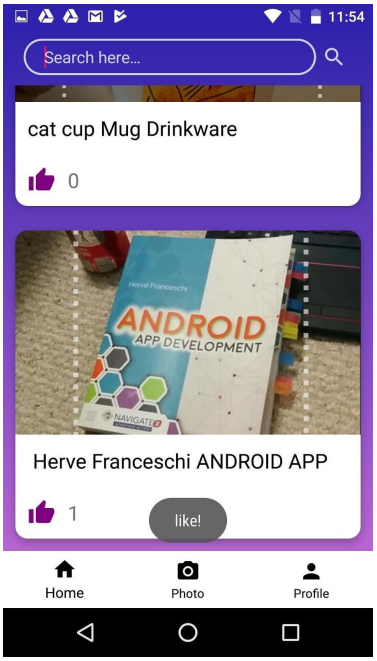



Figure 2.8.1 Like the post

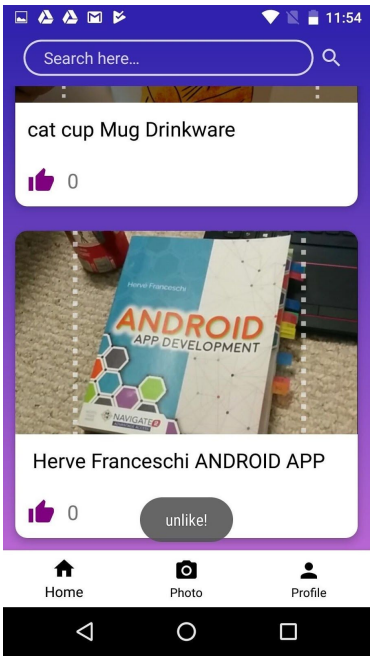



Figure 2.8.3 Unlike the post

RecycleView does not provide us the method like *setOnItemClick()* in ListView. Since we need two click actions here, one is to click the "thumb" to like others post, another is to click the post itself to see the details of a post, we wrote two interfaces in the adapter to realize the *onClick()* method and *onCheckedChanged()* method. Also, we need an enum class to distinguish whether the click action is on the specific post or the "thumb".

For the searching function, we reorder posts according to the search result. For each post, we count the number of tags that contain the search keyword and sort the posts in descending order by count. Then we add an update function to the adapter class to notify the dataset change and refresh the home page.

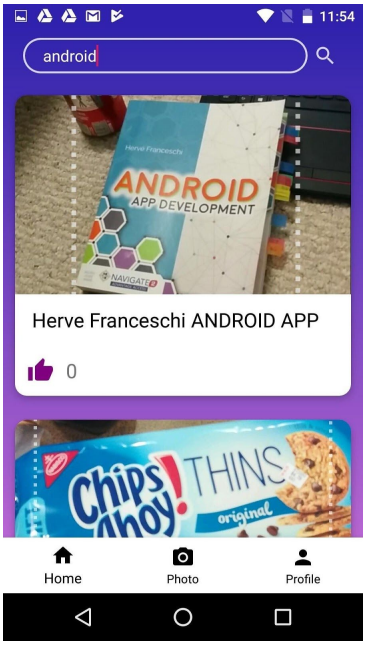



Figure 2.8.4 Search "android" screenshot

## 3. Future Work

### 3.1 Optimize the database structure

When the user is deleting a post, the current implementation of the database structure requires to iterate every user and find out whether the deleting post is under their favorite list. The performance will drop significantly when the post volume gets large. One possible solution is to make a separate table of the favorite list to store all the posts, and every post only stores the *userIds* that liked that post.

### 3.2 Improve tags filtering algorithm

Feature tags directly influence what the shopping items we will get. Currently, we did the duplicates removal, null tags filtering and only pick the high confidence score tags. We still can make the algorithm better by applying more tests.

### 3.3 Enrich user preferences

Within a limited time scope, we only have a few user preferences implemented in the current version. If we have more time, we will implement the recommendation system based on the tags being selected by the user. So the posts that most fit those tags will be displayed first on the user's home page. Similarly, we can do post filtering, avoid displaying the posts that users are not interested in.

## 4.Things to do differently

### 4.1 Database design

In the current version of our app, the factors we consider how to design a database schema are based on the convenience of reading and writing data. However, this will cause a lot of data redundancy, especially when our database stores a lot of image information. In this case, our database needs to be better designed and to be more robust.

### 4.2 Front camera

In our final version app, we don't have the front camera function. We faced compatibility problems when we tried to implement this part. In the old version, the front camera only worked well on old versions SDK, like SDK18. We didn't solve this problem so we decided to remove this function at last. Although it is not a very important function it's a basic function for a camera.

### 4.3 Cropping with rotation

Since all the tags obtained by image recognition are based on the image received from the image cropping section, the quality of the image is vital. Good quality not only requires the clarity of the picture but also emphasizes the validity of the picture. In the final version of our app, the recognition result shows better when matching the cropping frame to take a photo than taking a photo at a rotating angle. If we can accomplish rotation function at the cropping step, we can improve the accuracy of recognition results.

## 5. Weekly Blog

**week 2/24-3/02**: Brainstorming and team voted the project idea of Treasure Hunter.
**week 3/03-3/09**: Divided the mini research tasks and sent the approval email to the professor.
**week 3/10-3/16**: Break week, started working on the mini research tasks. We have advised by the professor and choose to stick with our original idea of image recognition task.
**week 3/17-3/23**: Midterm week, continue worked on the mini research task and prepared for the mini research task presentation.
**week 3/24-3/30**: Designed the App layouts and visualized with the storyboards.
**week 3/31-4/06**: Presented mini research tasks in class and started merging the main functions of the App.
**week 4/07-4/13**: Merged different login systems, also merged image recognition and keywords shopping item search parts. Accepted the suggestion from the professor of implementing image cropping feature.
**week 4/14-4/20**: Merged all parts as a single App, polished the App and prepared for the final presentation.
**week 4/21-5/02**: Final presentation, finalized documents and cleaned up the codes.

## Reference


1. https://developer.android.com/reference/android/hardware/Camera.html
2. https://www.instagram.com/developer/
3. https://developers.google.com/identity/sign-in/android/
4. https://www.youtube.com/watch?v=EM2x33g4syY
5. https://www.youtube.com/watch?v=oFsZKFW6YmM
6. https://cloud.google.com/vision/docs/
7. https://developers.google.com/resources/api-libraries/documentation/vision/v1/java/latest/com/google/api/services/vision/v1/model/package-summary.html
8. https://code.tutsplus.com/tutorials/how-to-use-the-google-cloud-vision-api-in-android-apps--cms-29009
9. https://serpapi.com/#integrations
10. https://www.youtube.com/watch?v=a6u3d2ssyjY&t=939s
11. https://www.youtube.com/watch?v=LOcD1evBcSA