Map of Photographers

Android app created to introduce photographers to new scenic spots.

Jooyeon Yang - jy24328
Michael Chang - mlc5457

1. Introduction

When we travel to a new city, we are always excited by popular restaurants and cafe there. Besides food, a hidden photo spot also inspires us! We can also have fun while searching and walking to the location which is shared by others. Also imagine that you open our app in the city you're living in for years, and just found that there're still locations you've never been to (maybe they are places you walk by everyday but never pay attention to).

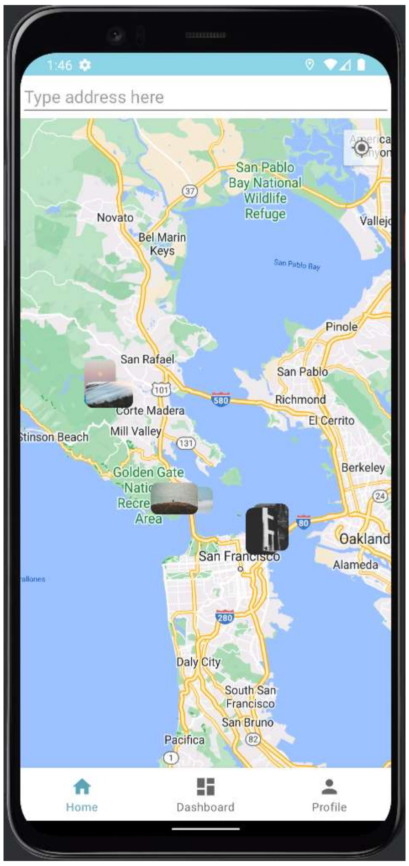
That's the reason we are going to launch our app, which provides an intuitive platform for users to upload / manage / browse photo based on geographical information, and allows users to be inspired by photo spots or re-discover their cities.

Users are able to share their photos with a brief description. Once they're uploaded, geographical infromation will be extracted from EXIF data and the photo will be pinned into the map. Other metadata such as focal length, shutter speed, etc. will also be recorded and shown once user clicks on that photo.

Users can also browse photos shared by others, and like their photos. They can also search photos by either user name or camera setting. For example, if you only bring your ultra-wide camera lens while traveling, you could find photos by searching `16mm` in Dashboard.

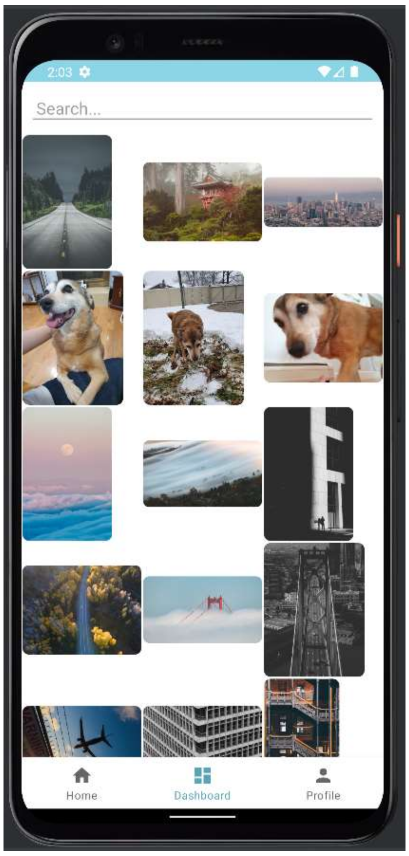
2. Screenshots

2.1 Home



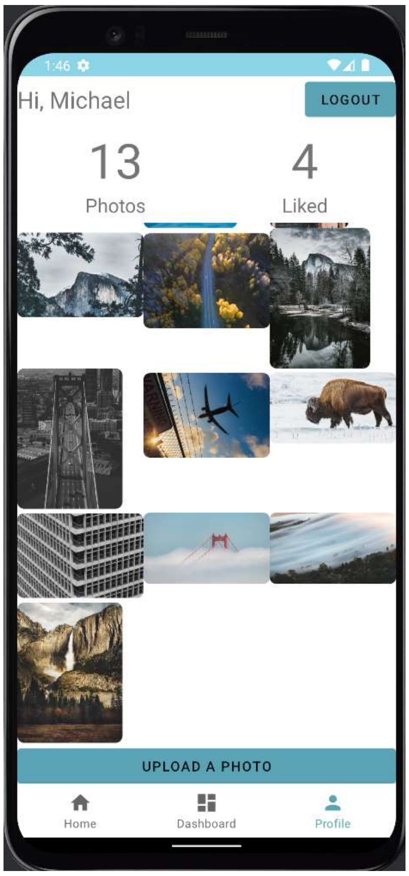
A map embedded with public photo markers on it. Photo markers will be resized when zooming in or out, to avoid unreasonable image size shown in the map. When a user click a image, an activity with image details is opened (see section Activity_Single_Image).

2.2 Dashboard



Here in Dashboard, users can browsing public photos (in upload timestamp order, the most recent one is shown first). There is a search bar, and users can search anything like title (e.g., `doggo`) or exposure time (e.g., `10s`).

2.3 Profile



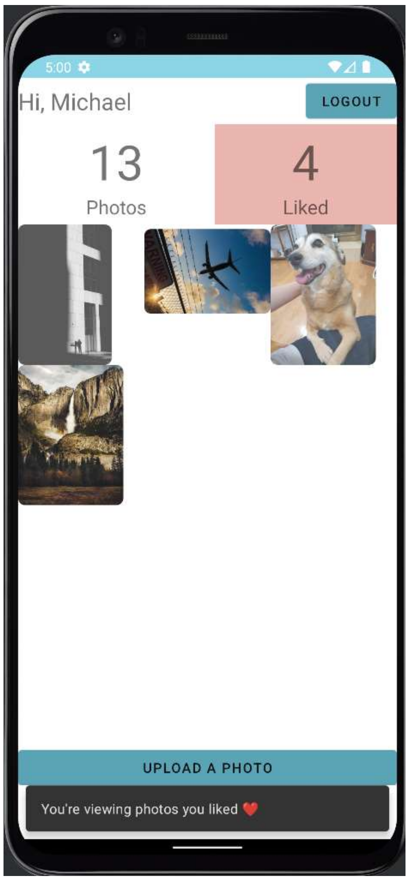
Firebase Authentication

Here user can login through Firebase Authentication and manage their photos. After login, user can click their names (right after `Hi,`) to change it. To save the name change, either press enter or click the input field area. This display name will be shown in the details page when viewing a photo.

Upload Photo

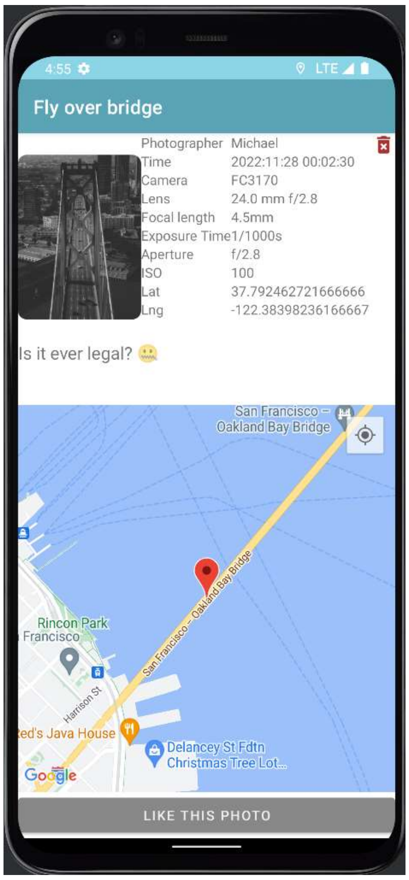
At the bottom there is a upload button, and users can upload existing photo through it. After selecting a photo, exif data such as shutter speed, camera model will be extracted and shown. Users will have to provide title (required field) and description. In addition, if there's GPS coordination, a marker will be placed in the embedded Google Maps. Otherwise, users have to click to place a marker in order to submit their photos. Users can set the photo privacy either private (only seen by owner) or public. After submitting the photos, users will be redirected back to profile page.

Liked Photo



There a text field showing how many photo a user has liked. By clicking the text field, the number is highlighted and users can browse photos they liked. Users can back to normal mode (viewing photos they uploaded) by click the text field again.

2.X Activity_Single_Image



Photographer	Michael
Time	2022:11:28 00:02:30
Camera	FC3170
Lens	24.0 mm f/2.8
Focal length	4.5mm
Exposure Time	1/1000s
Aperture	f/2.8
ISO	100
Lat	37.792462721666666
Lng	-122.38398236166667

Is it ever legal? 😬

Here users can view the details and geo-location of a photo. At the button users can like / unlike a photo. By clicking the thumbnail, users can view the photo in full-screen mode. If user is the owner of photo, a trashcan icon will be shown and user is able to delete the photo.

3. API Used / Third party libraries

RecyclerView

RecyclerView is used in Dashboard and Profile fragments, for showing photos in grid layouts.

Firebase Authentication

Firebase Authentication is used for user login / logout. After user login, they have unique `uid` and can upload / like photos, and view their private photos. They can also delete their own photos after uploading (by compare `ownerUid` of photo and their `uid`).

Cloud Storage for Firebase

We use Google Cloud Storage for managing uploaded photos.

Cloud Firestore

We use Google Cloud Firestore Database for managing detailed information of photos, and who liked the photos (as an array of `uid` in each photo).

Google Maps

Google Maps are embedded in several fragments / activities in our project. We use it to show the location where a photo was taken, pinning markers on it, and also for Geocoding.

Glide

We use glide for downloading and rendering photo stored in Google Cloud Storage. To show photos as markers in Maps, we also called Glide to get `Bitmap` (instead of rendering into image view) by using `get()` .

EXIF

This is a key library we used in this project. Basically it will extract detailed information (e.g., data captured, location, camera model, aperture, etc.) and we will save that in Firestore database. However, some of the returns are not in the format we usually use. For example, `TAG_SHUTTER_SPEED_VALUE` is in APEX format, and we have another function to covert it to exposure time (unit is second).

Parcelize

When viewing a single photo, it will create an activity and pass the information of a photo via intent. Instead of using a bunch of `getStringExtra` , we set our class `Parcelable` so that we can pass the instance via `getParcelableExtra` once.

Reflect

We have search function in Dashboard fragment, and it simply searches anything matched in the photos. For example, you can search for either exposure time (e.g., `10s`) or the title of photo. To achieve that in a simple way, in `filterList` function we use `class.memberProperties` to iterate over all the member in a class (except for some sensitive data like uid) and search them.

4. UI/UX/display

Resizable markers

In Home fragment, we have a embedded Google Maps and markers with photos on it. Instead of getting unreasonable markers size (like blocking the whole map), we use `setOnCameraIdleListener` and check if zoom is changed. If so, we will re-draw all the markers based on the zooming value we have.

Full-screen view

When users are viewing a single photo, they can click on the thumbnail to enter full-screen view. Instead of getting overhead creating another fragment or activity, we used a hidden imageview and only shows it when user click on the thumbnail. It also helps to load the image in background (with Glide) when the activity is loaded.

5. Back-end and processing logic

Manage likes by users

Here a list of users who like a photo is integrated inside the document of photo. We use this way instead of creating another collection because when we are looking for photos a specific user (with `uid`) who likes, we can just use `whereArrayContains` to filter out. To add / remove a user into liked list, we use `arrayUnion` / `arrayRemove` .

6. Interesting thing we learned

We have leaned how to combine functions we implemented from flip classrooms and homework together and use them in our final project. It is also quite interesting to learn that we can `parcelize` a class and pass it in intent.

7. Difficult challenge

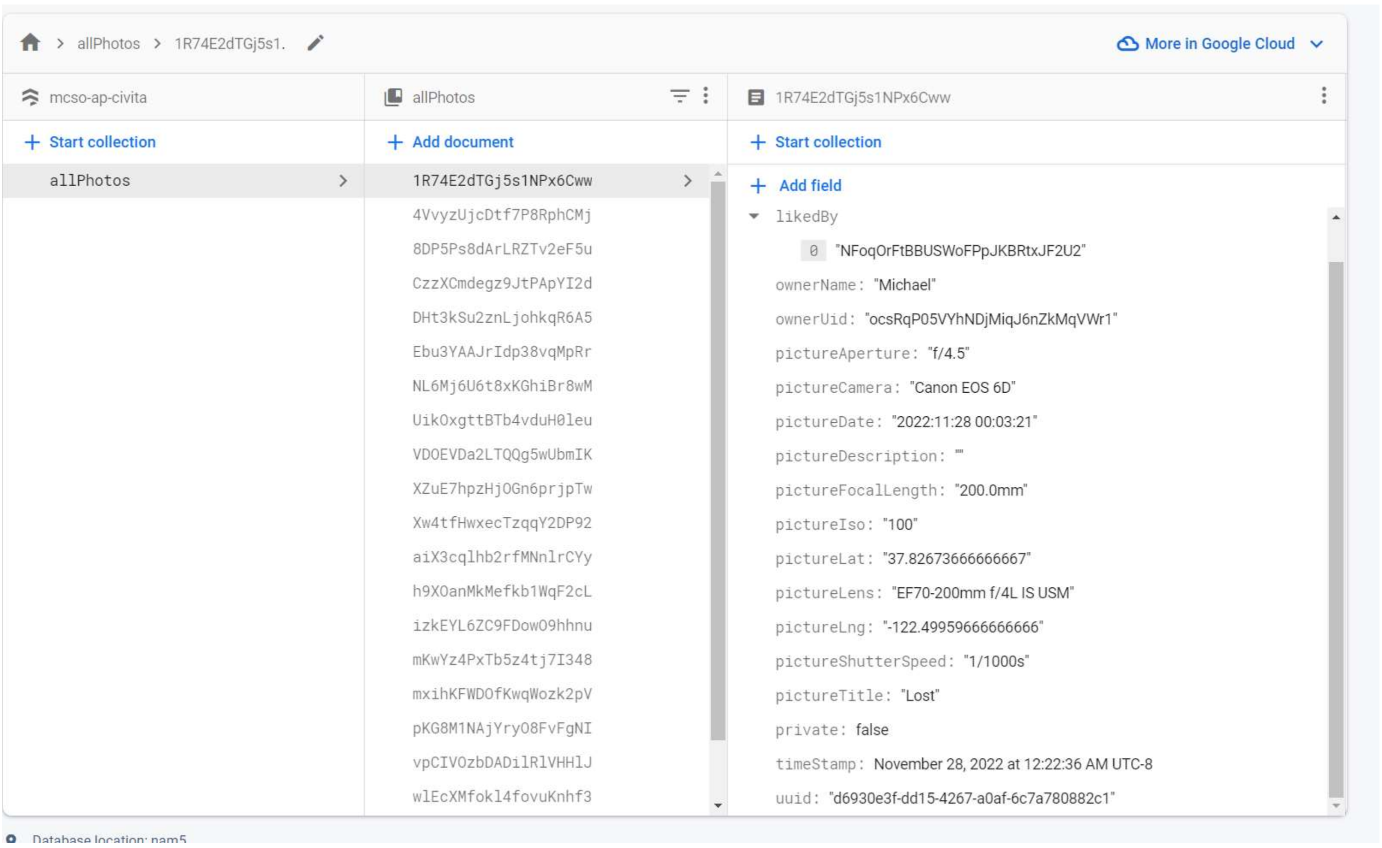
It is challenging for showing photo stored in Google Cloud Storage as markers in Google Maps with Glide because it's not like a image view. We've done searches and found that we can use Bitmap as a return type from Glide and render that in Maps.

We also found that the timing for refreshing list of RecyclerView is challenging because we use Activity for image uploading but the parent is a fragment. Instead of messing with `registerForActivityResult` , we simply overriding fun `onResume` in parent fragment. Each time the upload is completed and fragment is resumed, we refresh the list.

8. How to build / run our project

We have provided our `google-services.json` in the `/app` folder. You are able to build our project directly and run it in emulator or physical phone. For Firebase authentication, you can use `fake@example.com` with password `123456` or create your own account while login.

9. Database schema



allPhotos

```
1R74E2dTGj5s1NPx6Cww
4VvyzUjcDtf7P8RphCMj
8DP5Ps8dArLRZTv2eF5u
CzzXCmdegz9JtPApYI2d
DHt3kSu2znLjohkqR6A5
Ebu3YAAJrIdp38vqMpRr
NL6Mj6U6t8xKGhiBr8wM
UikOxgttBTb4vduH0leu
VDOEVDa2LTQQg5wUbmIK
XZuE7hpzHjOGn6prjpTw
Xw4tfHwxecTzqqY2DP92
aiX3cqlhb2rfMNnlrCYy
h9XOanMkMefkb1WqF2cL
izkEYL6ZC9FDow09hhnu
mKwYz4PxTb5z4tj7I348
mxihKFWDOfKwqWozk2pV
pKG8M1NAjYry08FvFgNI
vpCIVOzbDADilRlVHHlJ
wlEcXMfokl4fovuKnhf3
```

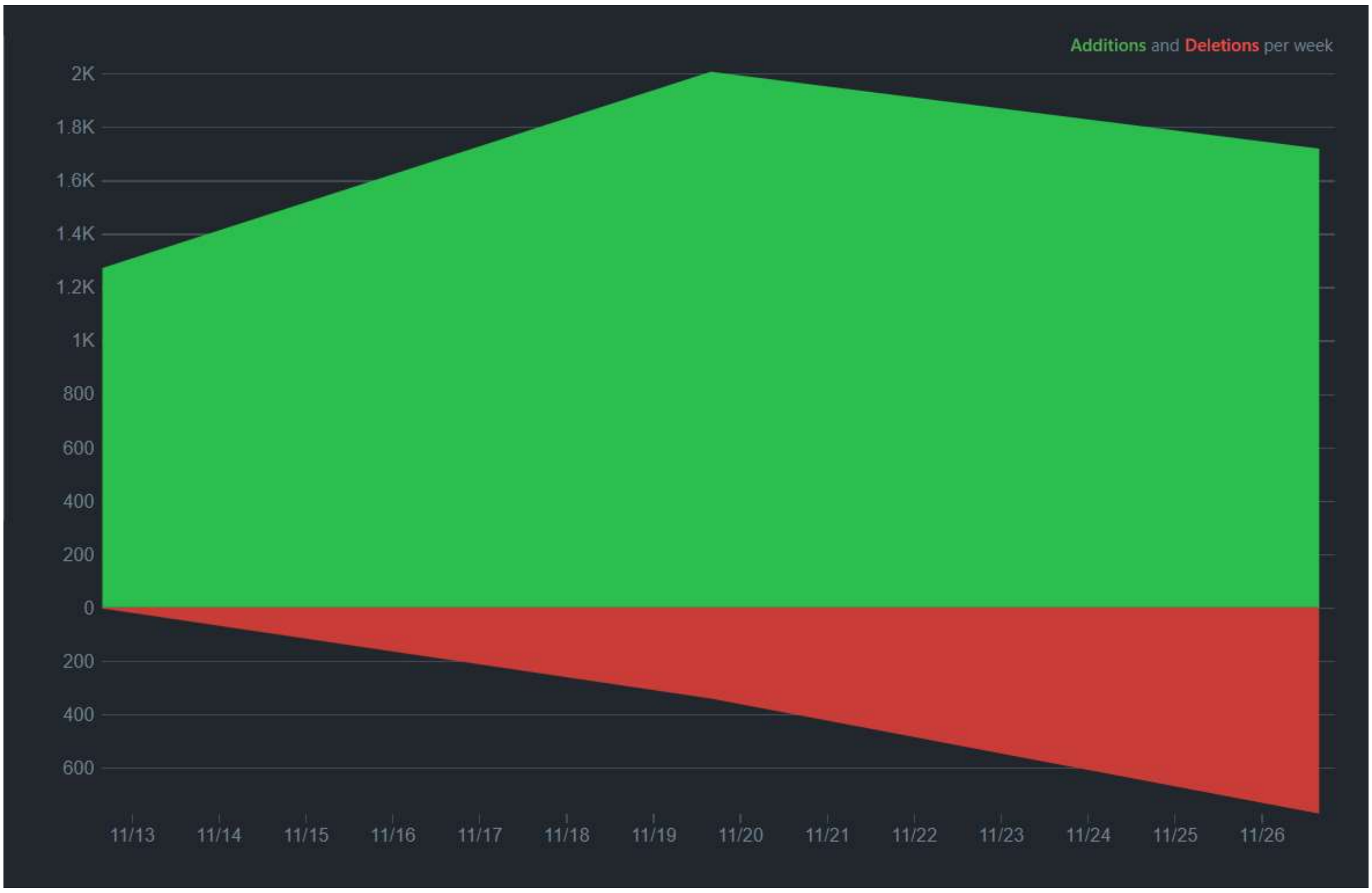
1R74E2dTGj5s1NPx6Cww

```
likedBy
    0 "NFoqOrFtBBUSWoFPpJKBRtxJF2U2"
ownerName: "Michael"
ownerUid: "ocsRqP05VYhNDjMiqJ6nZkMqVWr1"
pictureAperture: "f/4.5"
pictureCamera: "Canon EOS 6D"
pictureDate: "2022:11:28 00:03:21"
pictureDescription: ""
pictureFocalLength: "200.0mm"
pictureIso: "100"
pictureLat: "37.82673666666667"
pictureLens: "EF70-200mm f/4L IS USM"
pictureLng: "-122.49959666666666"
pictureShutterSpeed: "1/1000s"
pictureTitle: "Lost"
private: false
timeStamp: November 28, 2022 at 12:22:36 AM UTC-8
uuid: "d6930e3f-dd15-4267-a0af-6c7a780882c1"
```

10. Count of lines of code

- Kotlin: 1513
- XML: 1031

11. Code frequency graph


Additions and Deletions per week
2K
1.8K
1.6K
1.4K
1.2K
1K
11/13
11/14
11/15
11/16
11/17
11/18
11/19
11/20
11/21
11/22
11/23
11/24
11/25
11/26