Robert Nguyen

Kevin Overko

Xhejms Struga

**Geo Picture**

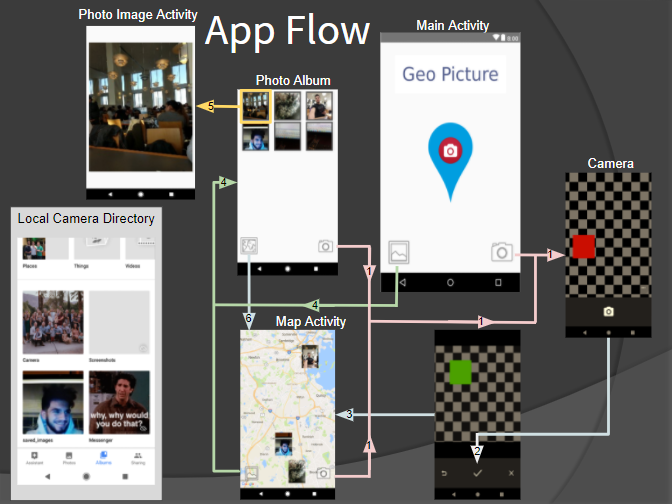
**Project Statement**

We created an app that targets all types of users who are avid or hobbying photographers. It motivates the user to explore parts of the world they haven’t seen before, as the location they captured the picture at are posted onto a map of the world. Same as any app, you would need a device to capture and save your memories on, like a smartphone or tablet with a camera and internal or external storage. In order to obtain the location of the photo, the device needs internet access with the GPS feature enabled which will then extract the coordinates. While our idea is an entire app, more established apps like Instagram and Snapchat have this as a smaller feature.

**Application Design**

We created the app functionality in the sequential order:

1. Camera intent
   * Save captured photo into new directory of phone storage
2. Map activity
   * Post bitmap onto Map activity
   * Retrieve GPS location/coordinates per photo
3. Photo Gallery activity
   * Retrieve photos from directory and post in Photo Gallery in GridView
4. Photo Image activity
   * View photo by clicking on it in the Photo Gallery



**Application Implementation and Evaluation**

**MainActivity.class**

The first problem we ran into was trying to implement the main Camera activity used to capture the pictures. While we wanted the camera to be built into our app with similar functionality as the default camera app (flash, zoom, front & back camera, etc), we ran into a myriad of problems that just caused the app to end up crashing. We ended up realizing this wasn’t the best approach and were making things very difficult on ourselves. When all we needed to do was access out camera via ACTION\_IMAGE\_CAPTURE. This solved a lot of headache if we wanted to add our own camera features stated above since the default camera app already had them.

Once a picture is taken, the user has the option to accept the picture or reject it and take another. If accepted, this class adds and saves the photo to a specific folder in the phone’s directory (root + “/saved\_images/”). If there is no such directory, then the folder is created. The image name is saved under the following format for easy distinguishing: MM-DD-YYYY-HH-MM-SS. We were unsuccessful with writing our own to grab the GPS location once the user accepts the photo, so we used code from the LocationUpdates files, as discussed in class, and modified it accordingly to our program. The GPS coordinates is then written into the .jpg file we created via the Exif file. We ran into a big problem while trying to setAttribute (Latitude and Longitude) on jpg file. It kept producing an error stating it required rational type, not string. After a bit of thinking we decided to simply store the latitude and longitude in a string and set it to the image’s description (we will later parse this in the MapClass file). This allowed for storage of photos to be kept within our phones storage and all that was needed was to pull from that specific directory, “saved\_images”.

**MapClass.class**

This class’s purpose is to extract the GPS coordinates from the image-discription of the .jpg file taken within MainActivity class and post the image on the map according to the coordinates. We create a PictureObject class that simply stores the bitmap image and the latitude and longitude coordinates for that image to create a simplistic yet organized format. We then create a PictureObject arraylist that stores each new picture. To fill up this arraylist we accessed the directory “saved\_images” and got the length of the folder. We iterated through the folder and got the bitmap and the lat long associated with each photo. As discussed in the MainActivity, we needed to parse the string “lat/long” into two doubles, so we created a function to do such a thing. Once the arraylist is created, we iteratively went through each member to display each object on the map, within the onMapReady function.

**PictureObject.class**

Simple object class that stores the bitmap image and latitude and longitude coordinates for use in the MapClass.

**PhotoGallery.class**

Initially we knew that we needed some sort of layout for the gallery of all our app’s photos. We all agreed that a grid view would look best but had no idea where to start. We created an imageReader that creates an arraylist and then we iteratively go through the directory path of the images and add all the images in that directory to the arraylist. Next we had to create an adapter that is the ‘bridge’ to the individual photo grid and the actual image. Each grid acted like a position, so each photo in the directory occupied a different position. But how did the program know how to differentiate one photo from another? We had to make sure each image had its own identifier, which is what setImageURI does. For each position an image is occupied in, the image is assigned a uniform resource identifier (URI) in string form. This maps the image to a URI for when the user wants to click on and view an image from the gallery.

**ViewImage.class**

This class simply views an image when clicked on while in the gallery. When an image in the gallery is clicked, this class pulls the URI that the image was mapped to and displays it in a bigger view.

Testing the app was quite simple as we did it per functionality and activity. We would test one feature first before moving onto the next, such as displaying the map, then displaying the bitmap on the map, then creating an object containing the bitmap and coordinates, and so on. Creating a fully functional picture object required first getting a bitmap and second getting the GPS coordinates. Majority of everything we did was sequential, except for the GPS coordinates which we struggled a bit on.

Our app is very simple, so we wanted to add to it with a clean and easy-to-use user interface. Getting from one activity to another is just a button away, which are located in the bottom left or right with the corresponding activity it is leading the user to.

The app is buggy so sometimes it will crash on a fresh startup, so we would have to delete the images in the directory and rerun the app.

One minor configuration the user may have to do is if they permissions check does not pop up, then the user must do the following: Settings>Find the app > Permissions > check off: Storage & Location.

One big issue we ran into with testing among the group mates is compatibility. More often than not, not everyone was running the same version of an API or SDK - so some functions would work for one person but not the other. This became very annoying.

A final problem we ran into was giving permissions to the user during runtime to enable the GPS and Storage. In the future we will try to resolve this error, but for now whenever the app is installed the user has to manually give permission to the app.

**References**

We used the LocationUpdates files that is posted on your course website. The primary use for these files was for obtaining the GPS coordinates which we could not write from scratch.

**Experiences and Thoughts**

Robert:

I thought the concept of the project was simple but little did I know that there would be so many minor parts that are needed to create a small feature. An example would be the adapter and imageReader in the PhotoGallery. Without them, you wouldn’t be able to connect the grid view with the image. While I did expect a lot of work to create an app, the amount of tools (functions for something specific) available to use was very overwhelming. There were many options to choose from and even so, I wasn’t sure which to choose.

Kevin:

This assignment helped me to take all that i had learned in the class this semester and apply it to an idea that we thought of as a team. This process took time and also a lot of updating/changing ideas. We were able to as a team get a nice clean base for a project that is very much upgradeable in the future.