GeoPic Final Report

Dave Becker, John Choi, Jonathan Nutter

I. Introduction

When visiting new places for the first time, a very common practice in the smartphone age is to take a picture of your surroundings. Have you ever wanted to take a picture of a location and wondered where the perfect place to take said picture is? Being able to see pictures that others in the same location have taken may allow you to find that perfect spot. Ever wondered what a place is like at different times of the day? Different times of the year? Being able to see photos taken at different times and dates in the same location may provide you with some idea of what it's like during those conditions. Whether you want the memories of the location, something that piqued your interest, or you just felt like taking a picture, it is likely others in the area have had similar feelings. Often, however, these other pictures are hidden away on a camera roll, or shared on social media where only friends and followers can see. That is where GeoPic comes in.

GeoPic is a photo-sharing social media application where users are able to share photos they take in an area, with those who are later in the same area. Users of the application see a map with pins all across the world where pictures have been taken and uploaded. However, you must be within 100 meters to view the uploaded photos. Users can also take photos of their own and add them to the map, by simply snapping a picture and submitting it. Each pin has a voting system allowing users to vote on their favorite photos. GeoPic can also be used as inspiration for your own photos. See a photo on GeoPic you want to replicate? The pin will give you an idea of where the photo was taken so that you can attempt to recreate the photo from the same spot. GeoPic can also help find hidden gems in your area. See a photo of some food at a restaurant that looks delectable? The pin's location will guide you to the restaurant where the photo was taken. GeoPic gives you the opportunity to look more into the area around you, that you may not have noticed previously.

II. Design Process

Once we had a good idea of how we wanted our app to function, the next step in the design process was for us to determine which classes and view controllers we would need. Our original class design can be seen in Figure 1 below:

Class	**Function**
GeoPicManager	Handles user location, Firebase, and adding pins
Location	Stores location and image associated with it

Image	Stores the image file

Figure 1: Original Design Classes

After we started coding, we heavily changed this structure. Due to the simplicity of our app, we chose to have a single class called “Pin”. The pin class has fields to store the pin ID, the URL of the image, the coordinate of the pin, the score of the pin (number of likes), the user ID who took the photo, the date it was taken, and an overlay for showing the radius of where the pin can be viewed. The pin class also has functions for incrementing and decrementing the score.

In our original design, we planned to have the view controllers shown below in Figure 2:

View Controller	**Function**
SettingsViewController	Handle user settings (changing name, password, etc.)
CameraViewController	Handle user uploading pictures
MainViewController	The map screen that shows the pins
PictureViewController	The view after selecting a pin
AuthenticationViewController	Handle user logging in and creating an account

Figure 2: Original View Controllers

This structure was slightly changed after we started coding. Rather than have an AuthenticationViewController, we realized that we needed to split it into LoginViewController and CreateAccountViewController. We also removed the CameraViewController because when we read the documentation for it, we realized that the camera view doesn’t need its own view controller and can instead be done in MainViewController.

Now that we had our view controllers and classes laid out, we needed to figure out the database schema. We knew that we were going to use Firebase [1] because it allows for rapid development, scalability, and provides a built-in authentication feature. Our original database schema was to have a user collection that is identified with a unique ID and contains the user’s email, password, and name. We also planned for a photo collection that was identified with a unique ID and contains the photo’s coordinates, a foreign key for the ID of the user that took the photo, and the score. Once we started working with Firebase, we realized that we had an incomplete understanding of how Firebase Authentication works. Each user in Firebase Authentication has a unique ID associated with it and you can access their email. However, to store other user information such as their name, we had to make a collection where the primary key is the user’s ID and the document has a field called “name.” We also had issues with our

plan for the photo schema. Firebase Firestore doesn't allow for image files to be used, so we had to upload images to Firebase Storage and store the generated URL in our photo collection. We also ran into some issues with allowing users to like photos. Having a simple counter for the number of likes doesn't account for remembering who liked which photos. Without remembering who liked which photos, we wouldn't be able to prevent people from liking photos multiple times. Thus, we added a subcollection to photos called "likes" that stores the user ID of anyone that has liked the photo. This left our schema looking like it does in Figure 3 shown below:

Collection	**Fields/Subcollections**
Photos	- date - location - photo_url - score - user - likes
Users	- name

Figure 3: Firebase Schema

III. Translating Design to Implementation

For our app implementation, our team decided to use Swift [6] with UIKit [8] and Xcode [7]. Because the app we are building is targeted for iOS, we did not have many options besides Xcode for our choice of IDE. Swift with UIKit, on the other hand, was chosen because it was a language and a framework that we were all familiar with and suits our implementation needs. Swift was chosen over Objective-C, which is an old official language used to build iOS apps before Swift was released. Choosing Swift over Objective-C was a no-brainer because Objective-C was replaced by Swift to satisfy the modern design languages that Swift provides in a more robust and easier way. For the GUI framework, our team also had to choose between UIKit and SwiftUI [9]. Ultimately, although SwiftUI is more modern and easy to use, our team decided to go ahead with using UIKit. There were two main reasons why we made this decision. First, because UIKit was around for more years than SwiftUI, all of the members were familiar with the framework and did not have to learn the new UI framework to start building. Because we only had a semester to complete this project, we thought using the UI framework that we all know was the right move. Secondly, although SwiftUI is deemed to be the future of iOS programming, professionals say that it is not quite ready for production yet [10]. SwiftUI is a framework that is going through a rapid change every year. Because of this, there are a lot of help posts online that do not use the latest syntax, and we wanted to avoid this problem by using the stable framework.

To speed up our implementation process and to eliminate tedious tasks, our team elected to use a number of appropriate open source libraries. As we were implementing the graphical

user interface, it came to our attention that the on-screen keyboard would sometimes cover the interactive elements, such as a text field. To remedy this issue, our team decided to use IQKeyboardManager [2], which is a library that would automatically refine the auto constraints of the screen to move everything that the keyboard is covering to the top of the keyboard, making the blocked elements accessible for interaction again. This library eliminated the need to manually computing the position of the keyboard and adjusting every element that is present on the view. The library also allowed the capabilities to only enable the feature on certain views, so our team ended up enabling it only on views that have multiple text entry fields. The second library we used was KeychainSwift [3], which makes storing the username and password on the device easy to implement and in a secure way. In GeoPic, our team applied this library to the login, where the user can use either Face ID or Touch ID (depending on the device) for login. Upon successful authentication, the device would pull the stored email and password from KeychainSwift and finish login using Firebase Auth. Again, this was a huge time-saver for our team, given that performing CRUD operations on Apple's Keychain requires some complicated programming. The third library that we used was Kingfisher [4], developed by onevcat on GitHub. This library allows easy downloading and caching of images from the web, and our team used this to download and display images asynchronously from Firebase. The last library that we used was SwiftMessages [5]. This library provides easy ways to display animated user-friendly alert messages. We integrated this library into our app to notify the users that they must be within 100 meters of the pin location and when the network is offline.

IV. Suggested Changes and Improvements

A suggested improvement for the design of the application is to improve the security of the database. The current setup of the database allows all users to both read and write to the entirety of the database. This flaw currently makes it possible for those with malicious intent to edit or delete data stored within the database. To fix this issue, it is suggested that more secure rules be used within the database restricting user access.

One main change that could help to improve the application would be improvements to the design of the user interface and the design of some elements within the app. The first screen a new user of the application sees is the login screen, which while functional, is very basic in its design. Some suggested improvements would be to add more color and imagery to the login page relating to the idea behind the application. This would give the application a more unique feel, and allow users to get a better idea of what they are potentially signing up for. The login process used could also be improved upon. Making users confirm their email address would help prevent spam accounts from being created. Additionally, providing users with more login in options such as through Google or Apple may increase sign up. Another beneficial change would be not forcing the user to log in every time the app is relaunched. The current design for the photo view and settings follow the login screen in that their designs are basic. More information about the taken photo, such as time taken and a category for the photo, would provide more insight into the

background of the photograph. Changes to the way information is presented and entered in the settings page could also improve the overall presentation of the application.

In addition to various design and UI changes, the addition of some supplementary features could make the design of the application more complete. Currently, when taking and uploading a photo the user is only able to utilize basic camera features such as flash and zoom. To improve the photo-taking experience for users, the addition of different photo modes and filters would be recommended. Also, adding post-processing capabilities to improve the quality of the photo after it has been captured and before upload would allow users to edit their photos to satisfaction before deciding whether or not to upload the photo. Post-processing features could include filters, cropping, changing orientation, or drawing on the photo, along with others. Another way to elevate posted photos would be to add a feature allowing the creator to add a caption to the photo describing it. Building upon this feature, a commenting feature allowing other users to comment on a photo post would be recommended. This would allow users to communicate with the original poster and ask questions relating to the photo. Adding a voting system to the comments would allow for the most popular comments to be the most commonly viewed. Adding different photo categories would allow users to better curate their experience within the application. Categories could include items such as food or landmarks and allow users to filter pins that they view by the selected categories. The filter could expand to include the date and time in which photos were taken for users to choose from. Another suggested feature that would elevate the application would be the addition of augmented reality capabilities to the application. Viewers would be able to view photos at their locations in augmented reality, being able to see previously taken photos and compare them directly to the present state of where the photo was taken.

Currently, the application only has a very small, controlled user base so many issues that may arise from a larger, uncontrolled user base have not been accounted for. One issue would be explicit or offensive imagery being uploaded to the map and removing it before it reaches the masses. One recommended improvement to combat this would be to add a report button to posts allowing users to report photos for various reasons. Photos with a high enough report rate would be removed from the map and reviewed before being deleted, or re-added. Another potential issue with a large user base would be the map becoming cluttered with pins, especially populous areas. One feature recommendation to potentially deal with this issue is to hide photos after they have been present for a certain amount of time. Additionally, when a cluster of pins is present, the posts could be combined into one larger pin where users can view posts within the radius.

V. References

[1] Firebase, Google, https://firebase.google.com/
[2] IQKeyboardManager, GitHub, https://github.com/hackiftekhar/IQKeyboardManager
[3] KeychainSwift, GitHub, https://github.com/evgenyneu/keychain-swift

[4] Kingfisher, GitHub, https://github.com/onevcat/Kingfisher
[5] SwiftMessages, GitHub, https://github.com/SwiftKickMobile/SwiftMessages
[6] Swift, Apple, https://developer.apple.com/swift/
[7] Xcode, Apple, https://developer.apple.com/xcode/
[8] UIKit, Apple, https://developer.apple.com/documentation/uikit
[9] SwiftUI, Apple, https://developer.apple.com/tutorials/swiftui
[10] SwiftUI Drawbacks: Why SwiftUI is not ready for production yet, https://www.iosapptemplates.com/blog/swiftui/swiftui-drawbacks