Time Capture Final Report

Dalton Meny

Project Advisor & Sponsor: Dr. Roberts, University of Evansville

Computer Science

University of Evansville

May 2, 2019

ABSTRACT

In today's world, it can be increasingly difficult to find pictures of events and places or how a certain place has changed over time. To solve this problem, this project has created an iOS application that allows users to share and view photos based on their location. This allows users to see a virtual timeline of the place they are at that point in time.

ACKNOWLEDGEMENTS

I would like to thank the project sponsor, Dr. Don Roberts for all his help and advice involving the design and creation of this project. I also would like to thank Dr. Deborah Hwang for her feedback and critique of my project throughout the entire process.

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

This project tackles an interesting problem: how to share pictures of a specific place with other people. There have been many attempts to solve this problem, but none have exactly tackled this problem in particular. To solve the problem, an iOS application has been created to allow users to take and share photos within the application and view other photos based on a radius from their current location. This section of the application has been designed to resemble a timeline that users can scroll through. This project used Ruby on Rails to create a backend database and took approximately 180 hours. To complete this project, there were many steps and pieces that had to be put together to achieve the final product

PROBLEM STATEMENT

Imagine that a person is walking down a street or down a path and they come across some landmark or a feature that stands out. They wonder how this feature has changed over time or who else has noticed it. They want to share this feature at this point in time with other people. Maybe an event is taking place that a person wants people who visit this location in the future to know about. In today's world, most platforms focus on the user first and the location second. If a user does not have a large social media circle, it can be hard to find pictures of the events and people around where they are. This is what makes this problem interesting. This project questions fundamental idea of what a platform revolves around. This problem requires that the location be the primary concern, rather than the user. This is what Time Capture intends to fix. Looking at photos of your current location should be a simple and fun event. It should not be burdensome or search-intensive, as it often tends to be. It should be simple to look at photos of your current location.

REQUIREMENTS

To solve the problem of wanting view more pictures of your surroundings, this project created an iOS application that will allow users to share photos based on their location. Photos are only viewable by a user if they are within a certain radius of where the photo was originally taken. This application is written using Swift. This is a language designed to be used for iOS, which makes it an ideal choice. Swift is used for the frontend. This project also used Ruby on Rails and MySQL for backend implementation. This backend is used to store the photos and their location. This is intended to be a simple solution to create a database.

SPECIFICATIONS

In this project, there are two fundamental actions. These are when a user takes and posts a picture and when a user views a picture. These separate functions are presented by the two main screens that will be the focus of the application. A third screen is focused on less important features such as general settings. For taking and posting photos, the application implements the following features.

- Only allow photos to be taken within the application
- Store the photo with geolocation of where the photo was taken and time the photos was taken in a database

These features allow the user to share their experience with others. People in the future will be able to view this specific point in time by a user taking a picture within the application. It will be possible to share a landmark at a certain point in time.

For viewing photos, the application performs the following features.

- Query the server for all photos within a certain radius of the user, based on their selection from a predefined list

- Display these photos in chronological order to the user
- Allow the user to scroll through the photos
- Display of photos should resemble a timeline.

These features allow users to see how a certain location or landmark has evolved over time. The application is able to present an easy way to view photos of a user's current location without much hassle. Viewing the photos should be enjoyable and not search-intensive.

Also, there are some smaller addition features. These include the following.

- Allowing users to see the photos they have previously posted
- Allowing users to flag content that is inappropriate
- Allow users to change general settings of the application

Overall, the design should be simple and straightforward. The UI should not be cluttered with extraneous features that add nothing to the final project. The application is divided into 3 main screens that each tackle the three sets of lists above.

DESIGN APPROACH

As mentioned before, this project consists of a frontend and a backend. The backend is used to store the photos, the location data, the time data, and the user information. The frontend communicates with the backend when the user uploads a photo or is looking at photos in their area. This radius in which users can view photos is variable based on the radius they select from a predefined list. The frontend is then responsible for displaying the photos. This relationship can be viewed in Figure 1.

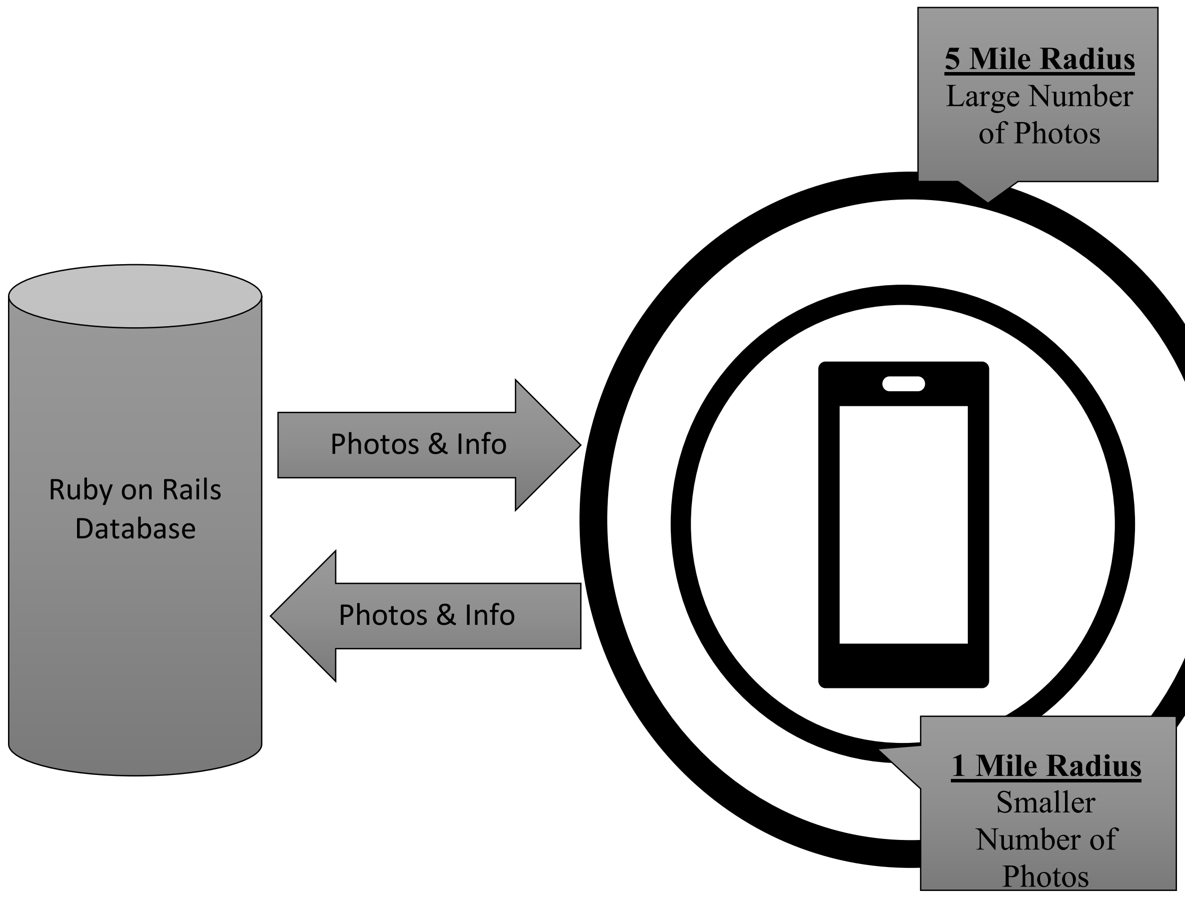


Figure 1: System Overview

Backend Design

The database consists of two main tables. The most important table contains the photo information. This includes the PhotoId, which is an integer unique identifier assigned to each photo uploaded to the service, Latitude and Longitude, which are doubles of the GPS data where the photo was taken, Time which stores the time in the yyyy-mm-dd hh:mm:ss.ff format, Path which is a string that give the path to the photo in the corresponding directory, and UserId which is an integer unique identification of the user who submitted the photo. This is needed for users to be able to see photos that they have posted. An example of this table is shown in Table 1.

PhotoId	**Latitude**	**Longitude**	**Time**	**Path**	**UserId**
0034	38.4789	-86.803	11/16/18 09:06:56	Folder/filename	12345
0035	37.9716	-86.803	11/16/18 22:48:01	Folder/filename	98765

Table 1: Table for Photo Information

The next table consists of the user's information. While this application does not allow users to add friends, it will give user controls over photos they have uploaded. A user may want to delete a photo off the application at a later time or may want to view the photos in the future. This table will consist of the UserId which was included in the last table, Email which is a string that is the user's email address/login, Name which is a string of the user's name, and password which is a string of the user's account password hashed. An example of this table is shown is Table 2.

UserId	**Email**	**Name**	**Password**
12345	dm242@evansville.edu	Dalton	abcdefg
987565	username@gmail.com	Dalton2	Adc123

Table 2: Table for User Information

The application communicates with the frontend through the use of APIs. These are run on a web server using rails. Using a get or post request, the application can perform the corresponding function to either get a JSON string from the API's or save new information to the database. This project utilizes SQLite through Ruby on Rails for the databases.

Frontend Design

The frontend consists of two sections for taking and viewing pictures respectively. These sections are represented by different tabs within the application. The first tab is for photo taking. This section is very straight forward. It allows users to take an in-app photo and then ask them for confirmation to upload it. A mockup of this section can be seen in Figure 2. After a photo is confirmed or canceled, it should return the user to the photo tab.

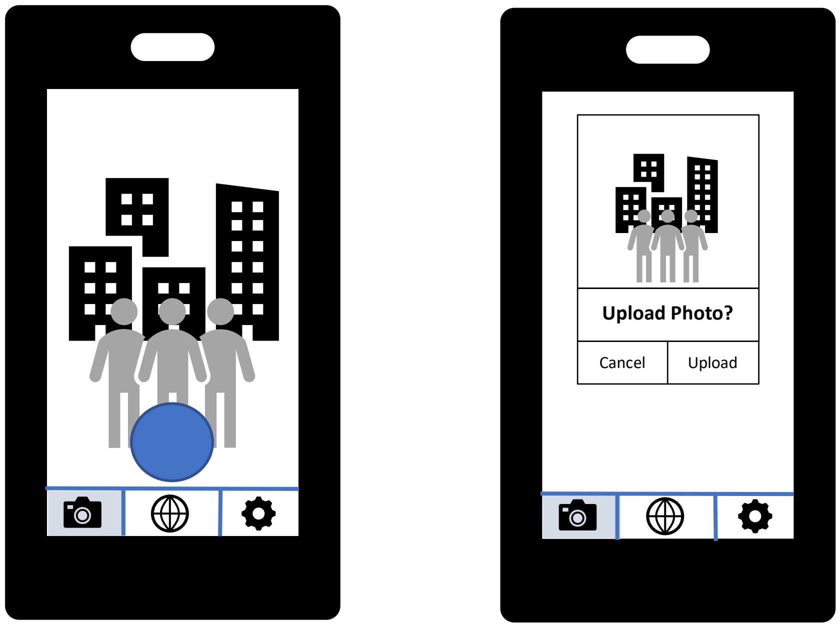


Figure 2: Taking and Confirming a Photo Mockup

The camera portion of this app was created by capturing the camera's output and displaying on the user's screen. This is possible using the AVFoundation framework. The capture architecture is shown if Figure 3. When the camera button is pressed it sends the user to the next screen. At this moment, the time and GPS location are recorded. The location is retrieved using the Core Location framework. Granting application permission through Core Location will show the user a pop-up box similar to the one in Figure 4. For confirming a photo, the user sees the photo taken, a save button, and a cancel button. The photo is sent from the previous screen, which cancel will take the user back to. When the save button is pressed, all the relevant information, included the previously saved time and location, is encoded into a JSON request and sent to the database. The photo is saved on a file system and the name of the path is saved in the database. This path's name is based on the time the photo is taken and the user's id number.

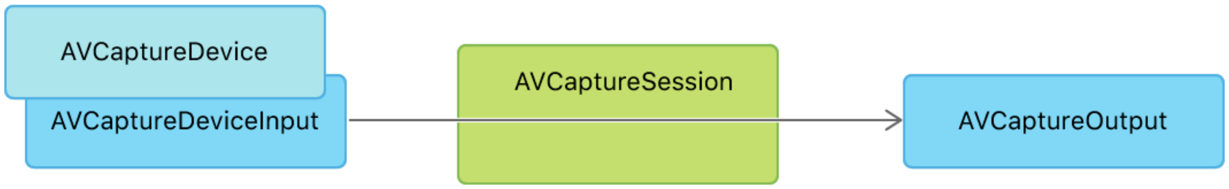
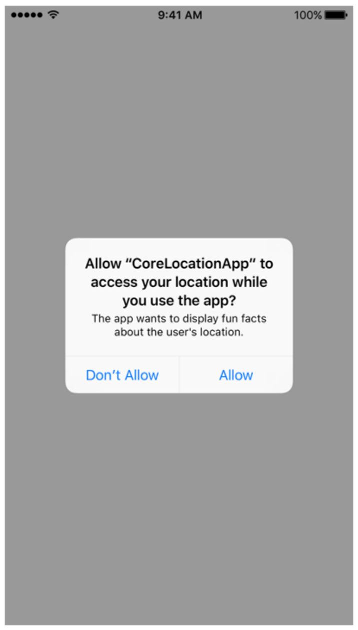


Figure 3: Capture Architecture[1]



Allow "CoreLocationApp" to access your location while you use the app?

The app wants to display fun facts about the user's location.

Figure 4: Requesting authorization to use location services[2]

The next tab in the application is by far the most interesting and most complicated. This is the section that allows users to view photos based on their locations. When a user first enters this section, they are prompted with a screen to select the radius that they wish to search. The options will be 500 feet, 1 mile, or 5 miles. A mockup of this design is demonstrated in Figure 5.

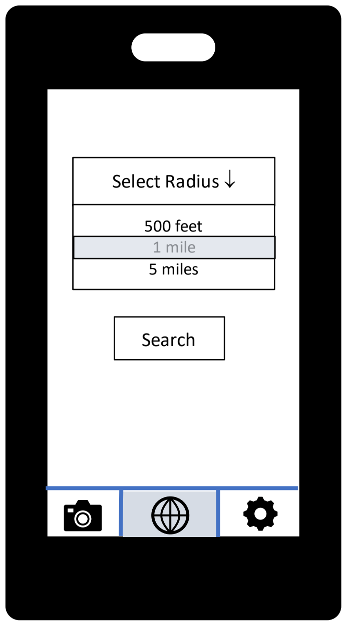


Figure 5: Select Search Radius Mockup

This screen's implementation was fairly straightforward. Each radius option is a button, that when pressed, determines which was pressed and send the conversion to meters to the next screen, which is the timeline screen. This is because the function to determine if a set of coordinates are in the current area requires meters.

After the user selects the radius that they wish to search, all photos that were taken within the selected radius of the user will be displayed. This is achieved by getting the photos from the database, decoding the JSON, and using the Core Location framework again. The CLCircularRegion class found in the Core Location framework which allows an application to create an object of this class and test a new set of coordinates and determines if it falls within the radius is used.[3] Using this, the selection of photos is refined and the photos are displayed in a timeline that the user can scroll through. Starting off, the oldest photo is displayed. Then users can scroll to the left to view newer photos. To the left and right of the main photo, the newer and older photos are displayed respective to their chronological order. This is possible using a

UICollection view to create a template for each cell and giving each cell at each index the appropriate information, namely the corresponding photos and the formatted datetime. At the top of the screen, the current radius is displayed and when pressed. This is possible by the previous screen passing the text of the button pressed to this screen.

The final tabbed section in the application is for general settings within the application. This section only has two actions for the user. The first is a "My Timeline" which sends the user to a modified version of the timeline screen where a value of -1 is sent to the screen. Doing this, all photos taken by the user are displayed regardless of their current location. The only option in the setting is for the user to logout. When this is pressed, the user is sent to the start screen.

The final screens not included in the tabbed section are for logging in and signing up. When a user first enters the application or logs out, they are sent to a start screen which simply displays the name of the application and two buttons – one to login and one to sign up. In the login screen, the user can enter an email and password. This screen gets the user info from the database, decodes it, then makes sure the entered criteria has a corresponding entry in the database. This sign up screen is slightly more complicated. The user must enter an email, name, and password. Firstly, this screen gets the current users from the database, decodes the JSON, and makes sure that the email is not already taken. If that is true, then the entered information is encoded into JSON and sends it to the API. On a successful login or signup, the user is sent to the camera screen. The user will remain logged in on a device until they logout.

Alternate Design

One alternate design this project faced is the ability for the current photo in the center of the screen would be enlarged. As the user would scroll through the timeline, the center photo would change, and a new photo would be enlarged. This feature is demonstrated in Figure 6 as a mockup. This feature was not implemented mainly for the complexity required to complete it, but also the dwindling time available to do so.

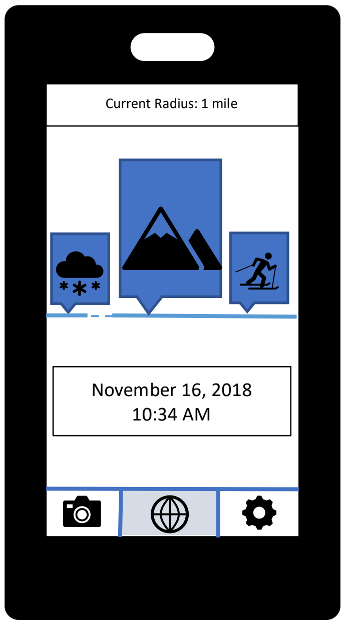


Figure 6: Timeline View Mockup

RESULTS

Overall, this project was successful in creating an application that is capable of sharing and viewing photos based on a user's location. Certain elements, such as the design could use some improvement, but the application fulfills the requirements of the project. The camera, confirmation, selection radius, and timeline screens are shown on Figure 7-10 respectively. The login and signup screen and shown on Figure 11 and Figure 12.

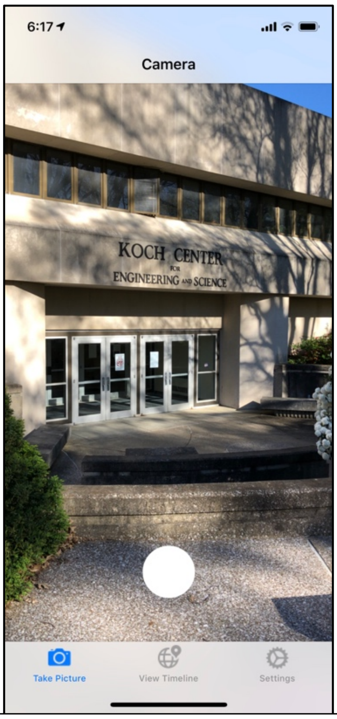


Figure 7: Capturing a Photo

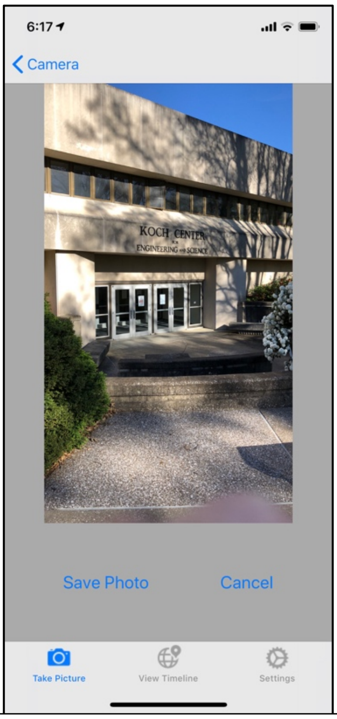


Figure 8: Confirming a Photo

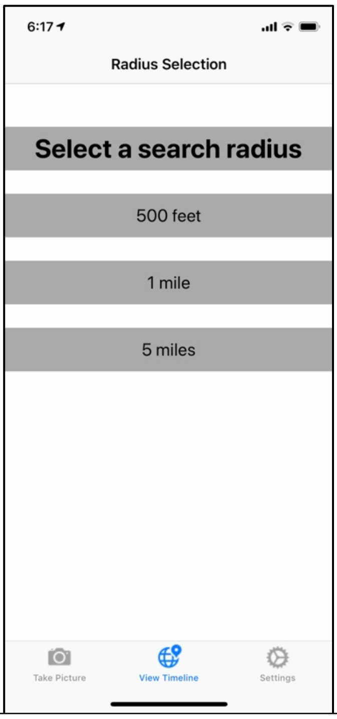


Figure 9: Radius Selection

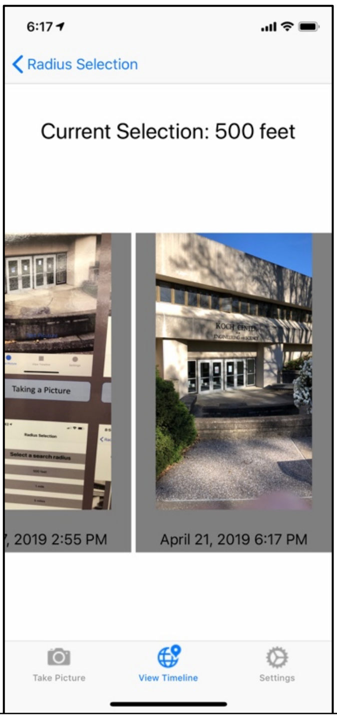


Figure 10: Timeline Screen

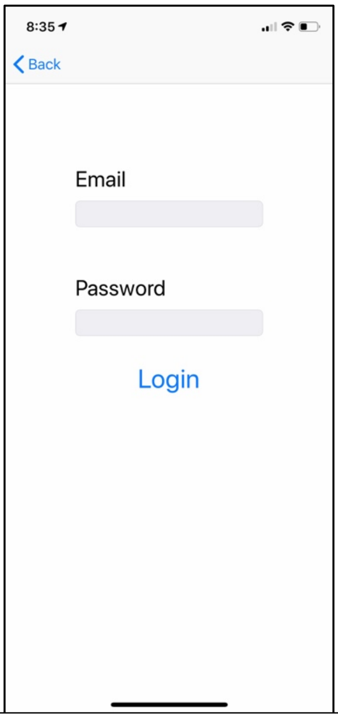


Figure 11: Login Screen

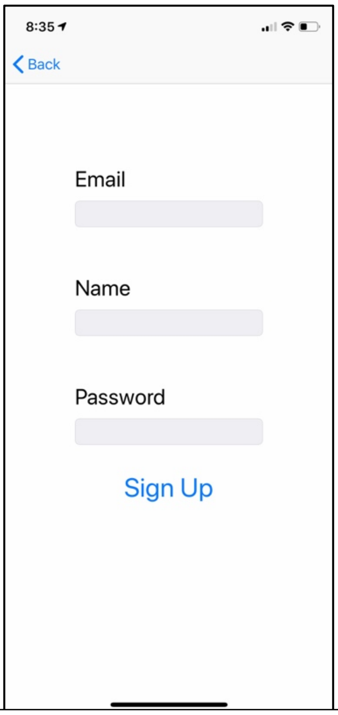


Figure 12: Signup Screen

CONCLUSION

In conclusion, this project has created an iOS application that is capable of sharing photos based on a user's location. It has successfully filled the major requirements such as the in-app camera and the timeline view to display pictures in a user's area. Doing this, as user is able to see a virtual timeline of their current location. The project consists of a frontend and a backend using Swift and Ruby on Rails respectively. These two sides communicate to share information and are critical for the functionality of the application. This application has required a large time investment and included a great deal of trial and error.

Overall, this project had a large learning curve. The projected started out very slow moving, but toward the end the project started making much more sense and seemed much simpler. Swift has many useful tools available, but it can be difficult knowing what to use. The APIs proved troublesome at first but quickly became trivial and did not require much work in the end. Making an application is a very interesting and fulfilling experience but it takes a great deal of hard work to accomplish.

Future Work

- Bettering API security
- Bettering password hashing
- Improve UI
- Add more features to timeline view
- Add more general settings

REFERENCES

1. "Cameras and Media Capture". *Apple Developer*. https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture.
2. "Core Location". *Apple Developer*. https://developer.apple.com/documentation/corelocation.
3. "CLCirclularRegion". *Apple Developer*. https://developer.apple.com/documentation/corelocation/clcircularregion.

BIOGRAPHY

The project engineer is Dalton Meny, a Computer Science student at the University of Evansville. Dalton is from Dubois, Indiana. After graduation, he hopes to get a job in Software Engineering, but is not sure where yet.