Interactive SC Historical Map Application by CapiStonsker

Joseph Cammarata
University of South Carolina - Columbia

James Davis
University of South Carolina - Columbia

Matt Duggan
University of South Carolina - Columbia

Lauren Hodges
University of South Carolina - Columbia

Ian Urton
University of South Carolina - Columbia

INTERACTIVE SC HISTORICAL MAP APPLICATION BY CAPISTONSKER

By

Joseph Cammarata
James Davis
Matt Duggan
Lauren Hodges
Ian Urton

Submitted in Partial Fulfillment
of the Requirements for
Graduation with Honors from the
South Carolina Honors College

May 2022

Approved:

[illegible]

Jose Vidal, Ph.D.
Director of Thesis

Steve Lynn, Dean
For South Carolina Honors College

Table of Contents

Thesis Summary

The Interactive SC Historical Map Application by CapiStonsker, hereby known as CapiStonsker, is an android application that creates a comprehensive user experience for individuals and groups to engage with their local Columbia history. Users of this application can discover local historical landmarks by scrolling through the interactive map on the home screen, searching for markers by name using the search bar, filtering by county, or scrolling through a list of markers sorted by proximity. If a marker catches a user's attention, he or she can tap it to learn more information, get directions, or save it for later by adding it to their personal wish list. Users can create an account to connect with friends and track their progress in visiting markers as well as saving their personal wish list. This application is useful for anyone who lives in or visits the state of South Carolina and wants to engage with their local community. The complex history of South Carolina is finally in a simple to use application that brings the states historical turning points and landmarks to the palm of your hand.

Introduction

Problem

The state of South Carolina has a depth of rich history dating back hundreds of years. Over time, many historical relevant locations have been marked with a plaque that provides historical information; however, there is no convenient way for any citizen to quickly and easily find the landmarks that are closest to them. Additionally, no interested person can easily filter through locations that they would find most interesting other than tediously looking through SC's historical archives. Furthermore, in the digital age that we live in, people are spending more and more time indoors and are engaging less with their local South Carolina communities than ever before. Our locals and tourists are removing themselves from the history by walking past these hardly noticeable plaques. It is some of these plaques that tell the story to which formed the diverse and interesting background of South Carolina we know today.

Solution

We have developed a mobile application that brings the local history of South Carolina to every smartphone in an engaging and compelling way, known as CapiStonsker named after our Capstone group. As a result of this project, anyone downloading our app can find nearby historical landmarks in a matter of minutes and people who love history now have a way to meaningfully engage with their local history. School teachers can now encourage with their students to go out and experience the rich history of our state for themselves through our friend feature. Through our plan route feature, families can find new routes to walk that will allow them to explore new parts of their community while learning something along the way. Everyone now has a reason to get out, go on a walk, and meet the people who might be able to tell a longer story than what is engraved on our markers.

Methods & Approach

Feature Overview

The following is a broad overview of features in the order that a user may encounter while using the app for the first time. See the Application Interface section for screenshots and a more detailed overview of each page.

When a user opens the application for the first time, a tutorial opens on the screen to show the user where to find the side menu and marker list navigation buttons as well as the search bar. On the home page, there is a map, centered on the user's current location, populated with nearby historical markers. There is a search bar at the top, which the user can use to display only markers whose name is a possible result for a search query, as well as, a filter button. The filter button can filter the map by county, the user's wish list, or the user's visited list. The user can use the bottom right button to navigate to the marker list page which is a list view of the home page. The list of the markers are sorted by proximity to the user. This page has the same search and filtering options as the home page and also has a button that can be used to open route navigation to a given marker. A user can also click on a marker on the home or marker list page to see more details and to add a marker to their personal wish list. From any page, the user can click on the button at the bottom left of the screen to open the side menu. From there, the user can log in or out and view the help page, which has frequently asked questions as well as a button to email app admins with any other questions or issues. There is also a plan route page, accessible from the side menu, that allows a user to plan a route to visit multiple markers. If a user is logged in, he or she can view and edit their account page, add or view friends, and see their visited markers and wish list by accessing the corresponding pages from the side menu. Logged in users can add a bio, display name, and profile picture (can take a picture or choose one from the camera roll) to their account.

Technologies

GitHub

We used GitHub as a version control platform to collaborate and document every stage of the app development process. In the beginning, we populated our wiki with planning documents, which include: 1) the ethical, security, and legal considerations, 2) detailed plans of the architecture, design, and requirements, and 3) user personas which illustrate specific examples to which our app would be helpful or critical to our users. Also, we each kept a personal log for every week of the project to show personal contributions and keep track of one another. As the semester continued and we began committing code for this project, we enforced branch protection rules, and utilized conventional pull requests for peer evaluations of each other's software. We assigned all project tasks through issues and created a pull request for the commits associated with each issue. This also allowed us to identify where bugs originated and undo problematic changes to our code by reversing the change from any pull request. Our GitHub issues track each detail of work we did for this project, who was assigned to which tasks, and who wrote each part of the code. We also have a detailed readme with instructions for deploying our app and a demo website, deployed using GitHub pages, that showcases the key features as well as the development process of our app.

Android Studio

We utilized the Android Studio IDE for the development of our app. We did all visual demos of our app using built-in Android Studio emulators, in particular we support the Pixel 2 emulator as described on our readme. We also connected GitHub version control to Android Studio and used terminal commands through the built-in Android Studio terminal for all git commands, such as branching, adds, commits, pulls, and pushes.

Google Firebase

We used Google Firebase for data storage of markers and user information. This includes user authentication. For user authentication, we only support account creation using an email and password, and only logged in users can add markers to their wish list, add friends, and access the account and friends pages. We also store user-specific information such as display name, profile picture, and bio on firebase for each user. The FAQs on the help page are stored and pulled from firebase, as are the markers with all of their information. All information on the markers was acquired from the South Carolina Historical Markers Program in the SC Department of Archives and History.

Flutter + Dart

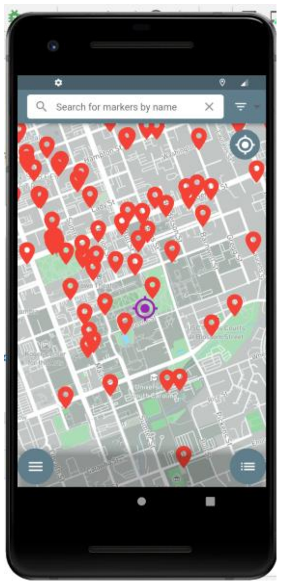
We chose to use Google's hybrid application framework, flutter, for the development of this application. Flutter uses the object-oriented language dart, so our project is built fully using this language. We took advantage of the hybrid nature of flutter to deploy natively in an Android environment while leaving the possibility of future iOS deployment or Web development. We also utilized flutter's fast run time and large supply of built-in libraries to build an efficient app and being able to add all desired features simply by using native libraries.

Map APIs (Mapbox, Flutter, Google)

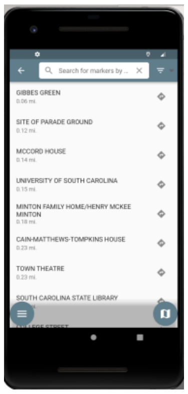
We used Mapbox with the Flutter Maps flutter package to create the map on our home screen and plan route page by building a custom map in Mapbox. This allowed our team to freely construct and integrate a map with the flutter maps library. Additionally, we were able to integrate routing information from the Mapbox APK. Similarly, we use the Google Maps to support route navigation to one marker (or up to 8 using the plan route page). If the user has Google Maps installed, we open the Google Maps application from our app for navigation.

Application Interface

Home Page



The home page opens on startup after the app briefly displays a splash screen with our logo and team name. The home page is initially centered on a user's current location (the purple icon in the center of the screen), but the user can scroll around and find other markers too. A user can always tap the button at the top right of the map to return to his or her current location. At the top of the home page is a search bar, which a user can use to search for a marker by name. This will filter the home page down to the markers which the query returns. The user can hit the "x" on the search bar or the backspace button on the keyboard to remove the query and return to the regular map page. Similarly, the user can use the filter icon to the right of the search bar to filter the map by county. If the user is logged in, he or she can also filter by visited markers or by their wish list. If a user taps a marker, it will open a marker preview pane at the bottom of the screen, from where a user can tap a star to add the marker to their wish list (if logged in) or can tap the pane to see the marker detail page. From the home screen, the user can tap the button on the bottom left to open the side menu (this option is available on all pages) or can tap the button on the bottom right to access the marker list page, which is a list view of the map on the home page (see "Marker List Page" below).



GIBBES GREEN
0.06 mi.

SITE OF PARADE GROUND
0.12 mi.

MCCORD HOUSE
0.14 mi.

UNIVERSITY OF SOUTH CAROLINA
0.15 mi.

MINTON FAMILY HOME/HENRY MCKEE MINTON
0.18 mi.

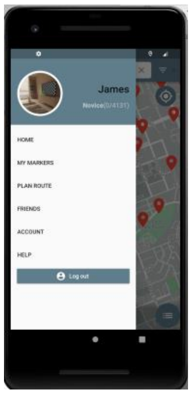
CAIN-MATTHEWS-TOMPKINS HOUSE
0.23 mi.

TOWN THEATRE
0.23 mi.

SOUTH CAROLINA STATE LIBRARY

Marker List Page

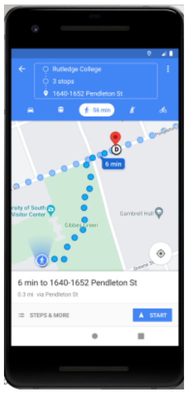
The marker list page is a list view of the home page. It has the same marker information, searching capability, and filtering, but markers are showed as a list, sorted by distance from the user's current location. The marker list page also has a button next to each marker that can be used to open in-app route navigation. The back arrow can be used to get back to the home page, and the button on the bottom right can also be used to get back to the home page.



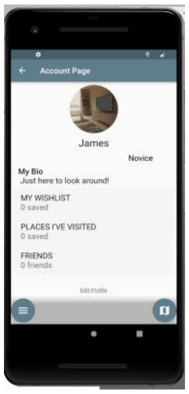
Side Menu

The side menu is accessible from every page via a button on the bottom left corner of each page. The side menu allows a user to log in or out and access the home, my markers, plan route, friends, account, and help pages.

Route Navigation



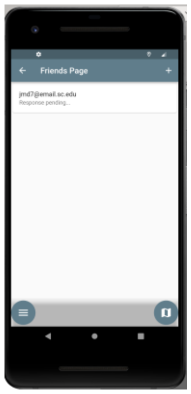
Route navigation is accessible from the marker list page and uses the Google Maps app to provide step-by-step directions to any marker. For a while, our only solution to this issue was to use a buggy and imperfect implementation of route navigation using the Mapbox API, but one of the last features we added is the ability to send coordinates to Google Maps to navigate to between 1 and 8 different markers.



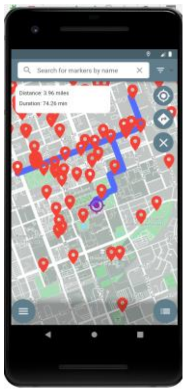
Account Page

For users who have created an account and logged in, the account page shows the user's display name (or email if no display name has been specified), profile picture, and bio (all of which can be edited on the edit account page, which is accessible via the link at the bottom of this page). This information is visible on the side menu as well for users who are logged in. From this page, users can also view their wish list, visited place, and friends, as well as the number of items in each category. Using the buttons on the page, users can access the side menu via the button on the bottom left and the home page via the button on the bottom right.

Friends Page



The friends page allows users to add friends, track pending friend requests, and track the activity of their friends. Once a friend request has been approved by both parties, this can be used to track each other's progress towards engaging with local history by visiting historical landmarks. From this page, a user can hit the "+" button to create a new friend request and can navigate to the side menu using the button by the bottom left as well as the home page using the button on the bottom right.

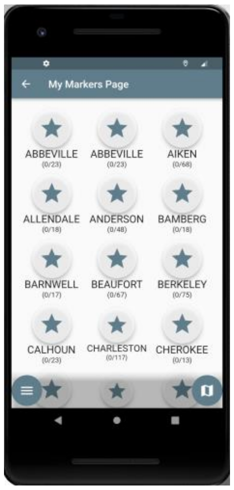


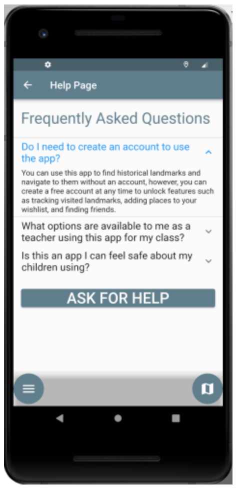
Plan Route Page

The plan route page builds on the home page to allow a user to plan a route to multiple different markers. A user can plan and save a route or clear the route by pressing the "x" button and starting over. This is useful for a variety of purposes, including planning a walk or road trip and trying to either maximize the number of markers visited or visit as many markers of interest as possible. The user can also create a route that visits up to 8 markers and use the button above the "x" to send a route to Google Maps and get step-by-step instructions. Otherwise, this page functions the same way as the home page and has the same buttons.

My Markers Page

This page tracks how many markers from each county a user visits and displays this information in an engaging way.





Help Page

Frequently Asked Questions

Do I need to create an account to use the app?

You can use this app to find historical landmarks and navigate to them without an account, however, you can create a free account at any time to unlock features such as tracking visited landmarks, adding places to your wishlist, and finding friends.

What options are available to me as a teacher using this app for my class?

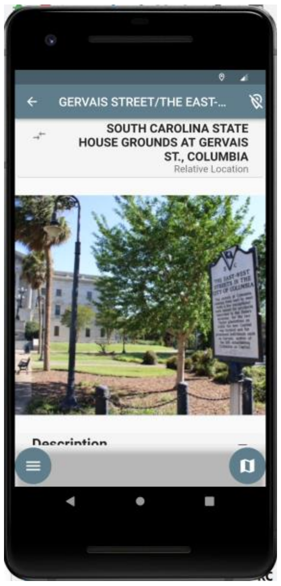
Is this an app I can feel safe about my children using?

ASK FOR HELP

Help Page

Finally, the help page provides a place where the user can go if there are bugs with the app or if a user is having trouble understanding the app for any reason or has any other questions. We have a tutorial that opens for first-time users, but the help page also exists to provide additional customer service. The first helpful feature is the frequently asked questions at the top of the page. These questions are backed up to firebase and are available to answer common concerns or issues associated with the app. There is also a help button that allows users to email concerns to a customer service email account that is monitored by app admins. If there are any additional questions or bugs we run into after the 1.0 Release, this page is a place we can post these issues so users are up-to-date. Lastly, from this page a user can navigate to the side menu or home page using the buttons on the bottom of the screen, as with the other pages.

Marker Detail Page


GERVAIS STREET/THE EAST-...
SOUTH CAROLINA STATE HOUSE GROUNDS AT GERVAIS ST., COLUMBIA
Relative Location
Description

This page can be accessed by clicking on any marker in list or map view and contains the name, address, description, and county of all markers. From this page, a user can add a marker to his or her personal wish list, see if they have visited this marker yet (via the icon on the top right, cross out if not yet visited), and learn more about the historical landmark by reading the description. We have added images to a few markers, as seen in the image to the right, and if we continue development of this app and decide to deploy it, we will add images to more markers. From this page, a user can go back to the previous page using the back arrow, access the side menu via the button on the bottom left, or return to the home page via the button on the bottom right.

Technical Obstacles

Implementing in-app route navigation

One of the biggest issues we encountered was implementing in-app route navigation. One of the developers on our team, Matt, took the lead here and ran into a lot of bugs using the Mapbox API and flutter maps plugin for this purpose. Both of these frameworks have libraries for this purpose, but the online examples have bugs in them and we were not able to find a solution that was bug-free and worked as smoothly as we wanted. We also considered redirecting the user to Google Maps (we considered Google Maps rather than Apple maps since we are deploying for Android and the tools we are using- flutter, dart, firebase- are all Google products). We ran into issues sending coordinates to Google Maps for a while and were also wary of fees for calling the Google Maps through intents in our app. We discounted this as a real possibility for a while, but in one of our final code commits for the 1.0 Release, Matt figured out how to send coordinates to the Google Maps application to support step-by-step route navigation for up to 8 markers. This is a major enhancement for our app and not only works far better than the Mapbox version did but also makes the plan route page more functional since we can send navigation instructions to Google Maps for up to 8 markers instead of simply letting the user create a hypothetical route but not being able to support directions, which was our best solution for the Release Candidate 1 release and all earlier editions of the app.

Selectively loading on-screen markers

Another major (and unsolved) issue with our app is our technique for loading the markers we need without slowing the app by loading every marker upon startup. One of our developers, Ian, worked on this and created the existing design but did not work on an altered design to change the way markers load. Currently, our app performs well, thanks in part to our generally efficient design and the fast speed of flutter, however, our app takes a while to compile and a relatively long time to load as well, and also has a high memory cost. Our app works well and could be deployed as is, but if we were going to sell our app on the Google play store (and possibly the Apple app store as well if we eventually deploy in iOS), it would probably be helpful to do some sort of smart marker loading where we only load markers that are on or near the screen, kind of like the way Google or Apple maps only load as much of the app as the user needs. This would be a helpful exploration topic to make our app more efficient and top-level production ready.

Implementing user authentication through external accounts

When we first began designing and planning the application for this project, we planned to implement user authentication through a user's Google account in addition to email and password authentication. We also considered adding other alternate authentication methods down the road. One of our developers, Lauren, took the lead on this and was not able to make Google authentication work. This does not affect the performance or other features of our application, but if we were to continue with this app and add updates beyond the 1.0 Release, adding user authentication via Google would be one of the first features we worked on.

Adding and storing profile photos

Another feature of the app that led to some bugs was adding and storing profile photos. Accessing the user's camera roll and camera were a little difficult, but the bigger challenges were to find a way to store images in Firebase and small, unexpected bugs that arose around the fringes of this part of the app. One of the developers on our team, Joe, took the lead in this area. He worked through the issues with implementing this feature, and by the time he had implemented it, the toughest bug to figure out was an issue where the picture wouldn't be displayed until a user left the account page and came back. This was one of several issues with in-app navigation we had to work out but is of particular interest because it came from an unexpected and seemingly unrelated area of adding profile photos. Joe ended up finding a solution to this bug, and we worked out all the issues with navigation, but it's worth noting that surprising navigation and reloading bugs are one area where our app ran into more unexpected bugs than many other areas of the app, which led to us learning a lot about debugging.

Testing and Validation

Unit Testing

For this project, we created a full suite of unit tests to test the accuracy of functions for our app (Ian, one of the developers on our team, took the lead here). This includes tests that check functions such as the function for marker distance and the search bar to ensure that given a known input, the output is correct. This prevents many logical bugs from being introduced to the code and remaining undetected. One thing we have realized as we wrap up this project is that we did not add unit tests until near the end instead of adding them continually and integrating automated testing more deeply into the development process. We did not create any tests for the proof of concept release, only created 2 unit tests prior to the beta release, and did not add additional unit tests until after all 1.0 Release code has been written and pushed to the main branch. While it is still helpful to have these automated unit tests, and we were still able to learn about testing through their creation and use, we have learned for future projects that it would be more helpful to do as our professor suggested and write tests for functions as we create them. This would allow us to catch bugs early and with less weight on manual testing, and would also allow us to require that all tests run correctly before commits are made to the main branch. So we ended up with a helpful suite of unit tests and learned about how we might improve our development process in the future by more deeply integrating tests from the beginning.

Behavioral Testing

We also created a full suite of behavioral tests for this application (Matt, one of the developers on our team, took the lead here). Like unit tests, behavioral tests are automated, however, instead of checking functions for correct output, behavioral tests simulate a user's interaction with the user interface to ensure that features are behaving as expected. We use behavioral tests to make sure tappable items such as the side menu, markers, and the various buttons on our application open or navigate to the correct place. Like with unit tests (see the section above), our implementation of behavioral tests was helpful in providing a layer of verification of our app's features beyond manual testing, however, similar to with unit tests, we learned several lessons about how we could have improved our process. Essentially, we determined that it would have been helpful to create tests as we went, primarily to provide verification of functionality with each code commit. This will be a helpful lesson for us as we begin to fine-tune our development habits as we begin our careers.

Manual Testing

As mentioned previously, manual testing was our primary method of finding bugs and improving this application throughout the development process. Each time one of us would pull from the main branch to update our Android Studio code base, we would open our app in an emulator to make sure the features we had most recently been working on were still functional. We would test this by using our application as if we were users of the app and trying to tap each button to make sure everything was working as expected. As developers, we improved in our technique and efficiency in doing this sort of manual testing as the class went on. One thing that was helpful to us was receiving instructor feedback on bugs and learning to look for issues with the code in the way our professor did. It was also helpful to simply learn by doing when it came to testing, and we found that our process of continuing to manually test became fairly effective by the end of the semester. However, we saw two problems with manual testing that highlight the need for automated behavioral and unit tests. First, as developers we can't see everything. Even when it came to our app that we were pretty familiar with, there were areas that our instructor and quality assurance team found issues that we had not considered. Adding unit test for each function and behavioral tests for each button or feature is helpful in catching more issues early on. Secondly, while we know our app well, not everyone who does quality assurance testing in a work environment will be familiar with the ins and outs of an application's source code. Therefore, it is helpful to have a full suite of tests that ensure the accuracy of each feature and give debugging teams a place to start, especially with a larger and more complex application.

Debugging

As with any software project, debugging was of major importance to us. As discussed above, our primary method for finding bugs was through manual testing. However, regardless of how we found an issue with our project, the debugging process was relatively similar. First, we worked to clearly identify a bug, to make sure it is reproducible, and to describe exactly what doesn't work. Then, we would create an issue (see GitHub workflow section below) and assign it to someone who would work on the bug. Once we isolated the problem feature or button, usually the person working on it would try to find the segments of code that could possibly be causing the error. Sometimes we would add a breakpoint to the code to allow us to utilize Android Studio's built-in debugging software. Once we got to this point, it became more art than science, and often we employed a strategy of trying to change different small pieces to see if that would fix the issue, looking online to see if others had similar problems, and finally talking to other developers on our team who had also worked on the code in question to bounce ideas off each other. This was an effective strategy for us, and many of these principles would apply well to the workplace. The most obvious way we can see to grow in this area (beyond integrating more automated tests as discussed above) is to utilize built-in debugging software more.

GitHub Workflow

As discussed in the GitHub section on page 4, we used GitHub for all planning and design components of this project in addition to all of our delegation and code commits. When it comes to our software development process specifically, we used the gitflow method as recommended by the professor. Using this method, we create an issue for each specific bug or feature and assigned it to a member of our development team. That person takes the lead for adding the feature or solving the bug, making commits as applicable. That developer also creates a new branch, specifically for the issue in question, making all commits to that branch. Then, once the issue is resolved, the developer creates a single pull request to merge the changes with the main branch, which links to and closes the original issue. We have branch protection rules in place so another developer must review and approve the changes before they can be merged. Using this process, we were easily able to assign work as a team and track who is working on each part of the application at any given time. This also allows us to visualize what work needs to be done for each milestone and how our progress looks over time.

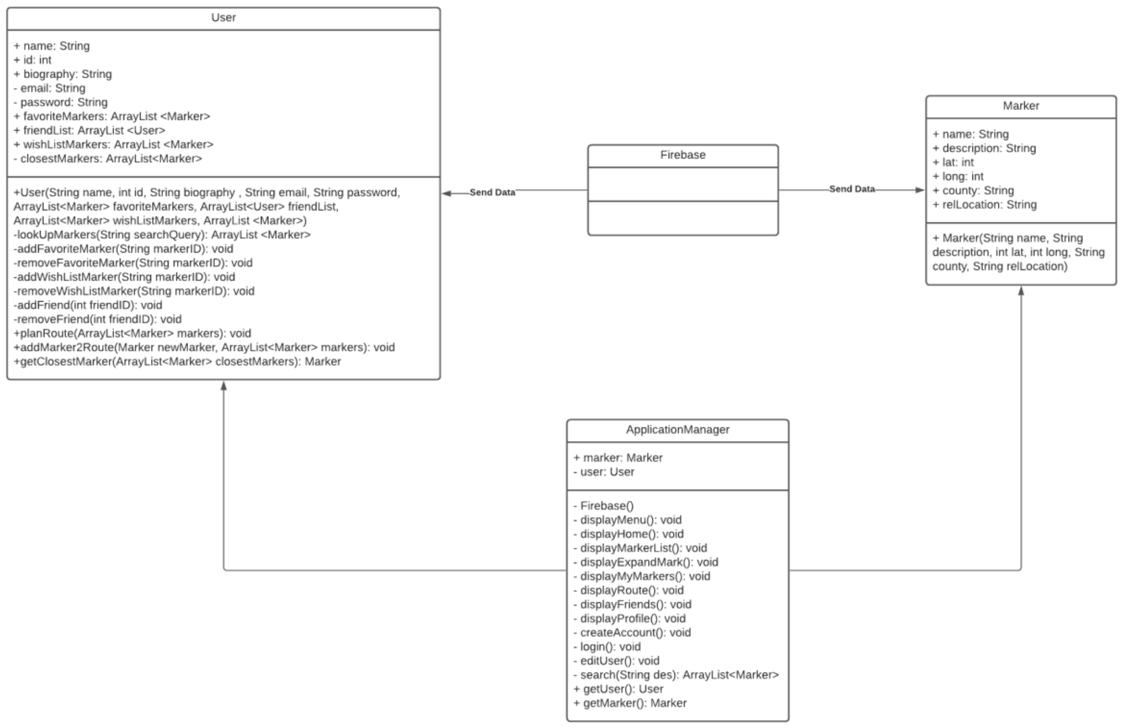
Conclusions

Developing this application was an enjoyable and learning experience for our entire development team. Most of us had never used GitHub to its fullest extent before, including using git command line commands, pushing to and pulling from main collaboratively, and assigning tasks via issues. Our experience with the gitflow method will be invaluable to us as we begin careers in software development and related fields. We also learned to interact with technologies that were new to us, including APIs, a programming language in dart, and other tools and technologies. We became comfortable with mobile app development in Android Studio, with deploying a release version of an application, with interacting with Google Maps, Mapbox, firestore, fire_auth, and other APIs. We learned to implement search and sorting functions, both on a map and on a list, inside of a dart application. Through all of this, we learned the importance of teamwork, as we leaned on each other to divide work on this application and maximize our productivity and capability. We learned to work through bugs in our code and learn to work with new libraries on the fly. As a result of this project, our entire development team feels more comfortable than ever working on a team to develop meaningful and useful software.

As for the application itself, having the opportunity to work on an interesting and useful application made this project much more enjoyable and worthwhile. This highlighted the importance of working on something we care about, as it was far easier to work through a bug late at night knowing we were developing an application that not only we could use, but also that our friends and family could as well. This was the perfect project to pay tribute to the state that we have called home during our time in college as we wrap up our undergraduate experiences and look ahead to what is next. This project has been an invaluable and fun experience that will set us up well to continue to grow and learn in a fast-paced environment and will always be something we will look back on fondly.

Appendix

Database Schema


User
+ name: String
+ id: int
+ biography: String
- email: String
- password: String
+ favoriteMarkers: ArrayList <Marker>
+ friendList: ArrayList <User>
+ wishListMarkers: ArrayList <Marker>
- closestMarkers: ArrayList<Marker>
+User(String name, int id, String biography , String email, String password, ArrayList<Marker> favoriteMarkers, ArrayList<User> friendList, ArrayList<Marker> wishListMarkers, ArrayList <Marker>)
-lookUpMarkers(String searchQuery): ArrayList <Marker>
-addFavoriteMarker(String markerID): void
-removeFavoriteMarker(String markerID): void
-addWishListMarker(String markerID): void
-removeWishListMarker(String markerID): void
-addFriend(int friendID): void
-removeFriend(int friendID): void
+planRoute(ArrayList<Marker> markers): void
+addMarker2Route(Marker newMarker, ArrayList<Marker> markers): void
+getClosestMarker(ArrayList<Marker> closestMarkers): Marker
Send Data
Firebase
Send Data
Marker
+ name: String
+ description: String
+ lat: int
+ long: int
+ county: String
+ relLocation: String
+ Marker(String name, String description, int lat, int long, String county, String relLocation)
ApplicationManager
+ marker: Marker
- user: User
- Firebase()
- displayMenu(): void
- displayHome(): void
- displayMarkerList(): void
- displayExpandMark(): void
- displayMyMarkers(): void
- displayRoute(): void
- displayFriends(): void
- displayProfile(): void
- createAccount(): void
- login(): void
- editUser(): void
- search(String des): ArrayList<Marker>
+ getUser(): User
+ getMarker(): Marker