CemetARy

Cemetery Augmented Reality

Group 2

Sean Grimes
Tevfik Koyun
Jordan Messler
Kyle Wu

Senior Design
11 - 29 - 2021

Sponsor
Dr. Amy Giroux, U.S. Department of Veterans Affairs,
National Cemetery Advisor

Special thanks to
Connie Harper

Table of Contents

1. Executive Summary

1.1 Abstract

This document serves as the initial project proposal for the CemetARy project, detailing the project's goals, requirements, function and future plan of action as well as the project members' personal motivations and interests. The CemetARy project aims to provide users travelling to cemeteries with GPS and augmented reality experiences, allowing greater ease of access to information pertaining to a cemetery while providing more casual users with a lightweight app that can accurately locate specific graves within a national cemetery. For administrators, the CemetARy private website gives a simple way of managing its database filled with thousands of veteran profiles from various cemeteries.

1.2 Introduction / Narrative Description

The United States national cemeteries have both historical significance as well as modern importance; first established in 1862 during the American Civil War, the United States National Cemetery system today consists of 171 cemeteries that mainly contain the graves of U.S. veterans and their family members. National cemeteries serve as both cemeteries as well as memorials honoring the sacrifices of the United States veterans, and people from around the world come to pay their respects at these national museums annually. The Arlington National Cemetery alone receives more than three million visitors each year.

Given that there is both significant traffic as well as a vast amount of veterans buried at these locations, it comes as no surprise that navigating these national cemeteries can be overwhelming for first-time visitors. Finding a particular veteran amongst the many identical tombstones can be a challenge, and finding information about that veteran's life can be even more arduous still.

The CemetARy project seeks to alleviate these problems by providing a lightweight app that can provide both GPS and Augmented Reality navigation to a given veteran's grave as well as provide information on that veteran's life. The app can swap between top-down 2D GPS and AR to make navigation to a

particular grave easier, and then can use Optical Character Recognition (OCR) to read the tombstone and prompt the user if they'd like more information about the veteran.

Our project seeks to improve the existing CemetARy project design. By providing a more robust administrative website, database, and application, the CemetARy project may very well be usable for the general public at any of the 171 national cemeteries in the United States.

2. Technical Content

2.1 Objectives

Our main goal is to take a previously developed application and upgrade its features to the specifications requested by our sponsor Dr. Amy Giroux. We will update the administrative web page to be able to handle more varied requests as well as the application to be user-friendly and up to par to modern standards. We will improve upon the application's features to be more accurate, and finally, we will create a new, robust, versatile database that can support the application.

2.2 Motivations and Personal Objectives

2.2.1. Sean Grimes

My personal motivation for working on this project is wanting to work on useful software that works outdoors using augmented reality. I would like to have more experience with both augmented reality and computer vision, as well as working with databases. This project provides opportunities for both skills to grow.

Another unique aspect of this project is that, rather than building the application ourselves, we are improving an application already built. One aspect of this that appealed to me is being able to work with code written by former students and building on that. I have not had much experience building on others' programs, so this project will allow me to gain more experience with this. Another aspect of this that is appealing to me is that we will be deploying an application that will be used in the real world by the National Cemetery Administration.

2.1.2. Tevfik Koyun

I chose this project because I really would like to help people.This project has different portions like website, database and the mobile app.Working on this project with many different versions,that's exciting to me. While I am creating a project to be useful to others, I would like to improve my team skills on a real project. We are going to use many features which I do not have much knowledge of currently, but I will learn these skills and use them practically in this project.

I would like to help people to find their national heroes. By taking the time to visit the cemetery where one's loved one is buried, one can pay his or her respects. My goal is to provide useful technology to find veterans, our heroes. Also, we do not know about many of them. With this app, when one scans the gravesites of those one is interested in, one can find their legendary biographies.

2.1.3. Jordan Messler

I was interested in this project on a career-side because I wanted to learn more about project management and the technology used in this project. Augmented reality software has an untapped potential that I believe will flourish in the next few years. More experience in frontend technology will diversify my expertise, and managing this project was invaluable in my growth as a programmer, where large projects and emphasis on team dynamics are commonplace.

On a personal side, I have seen the beauty of these places firsthand, and the incredible honor given to the people who put their lives on the line. It is important to respect those lost, and to help those who want to pay respect to them. I believe CemetARy aids this idea by making cemeteries easier to navigate and have accessible information on the deceased. Knowing that this could help grieving people and researchers is a huge push, not to mention that through accessible sources like CemetARy, these veterans' legacies can live on.

2.1.4. Kyle Wu

This project appealed to me because I felt that there was a particular need for it to be completed. Cemeteries are places of solemnity that are seldom visited frequently by the same people, and it can often be taxing to navigate through such an area for many reasons; families visiting through the area are expected to know where their dearly departed lie, but often friends of the family who wish to pay their respects have no such knowledge and find it difficult to ask. The departed deserve the utmost respect, and hopefully the application we build will help assuage some tension for users visiting cemeteries.

A secondary reason why the project appealed to me was the broad use of technologies at play in this application. The webpage runs on HTML and uses multiple APIs in order to communicate with the database. The database uses MySQL and additionally is hosted by a site that runs on PHP. And finally, the mobile application uses Unity as its engine and relies on Google Vision to provide OCR.

There are so many technologies to learn that the project is incredibly interesting from a wide variety of angles, if a bit demanding. I hope that I'll be able to glean a lot from the different fields that play a key role in this project.

2.3. Goals

The previous iteration of this project implemented the GPS, AR, and OCR technologies, but left some room for improvement. Firstly, curved text, while common on tombstones, presented a problem for the OCR, which our version seeks to fix. Furthermore, national cemeteries can be located in a number of places, many of which do not have good internet connection; our new version needs to be able to work without the internet. Additionally, the CemetARy application works on iPhones, but also needs to be able to work on iPads as well as potentially Windows Tablets. And finally, the database powering the CemetARy application needs to be more robust so as to be able to import spreadsheets and add and delete entire graveyards from the database at a time.

2.4 Broader Implications

While this project focuses its efforts primarily upon national cemeteries, it could potentially be used for any cemetery that has been catalogued in the same way. There are 171 national cemeteries that this application could currently apply to, and a countless number of users who could benefit from such an application. Though perhaps not commercially lucrative, this application could serve to help people pay their respects in the event that they find it difficult to navigate through a cemetery, or in the worst case, that they are not able to visit in person at all.

2.5 Legal, Ethical, and Privacy Issues

The use of GPS has long been contested ethically with little avail due to its undeniably invaluable utility. Though use of GPS in cemeteries may seem grounds for privacy issues, any cemetery that is open for visiting hours is considered public property and is thus viable for GPS usage. While we will be testing our GPS using drones, the cemeteries we will be visiting will additionally be cleared for drone usage. There are also no legal ramifications pertaining to the information provided by our project as all data logged into the national cemeteries is considered public knowledge.

We will be using APIs that the previous team has purchased for the project, and will be sure to review the terms and conditions of the use of APIs to ensure that we are still in the clear when using them for our iteration of the project.

3. Technical Goals

3.1. Technical Objectives

The main goal of this project was to improve upon the previous implementation. Every portion of the project worked but did not achieve desirable results.

3.1.1. OCR

The Optical Character Reader (or OCR) did well with standard text but did not work well on the irregular text many headstones have. Curved and protruding text drastically reduced the accuracy of the original OCR.

3.1.2. Offline Functionality

National cemeteries are located in a number of places, many of which do not have good internet connection. Our new version needed to be able to work without the internet. By having a user download a “cemetery pack,” they would be able to use relevant database information.

3.1.3. Mobile Additions

The original CemetARy application worked on cell phones but did not work on tablets due to scaling issues. We aimed to fix those issues. The mobile interface also needed to be improved to a more professional format.

3.1.4. Database Additions

The database powering the CemetARy application needs to be more robust so as to be able to import spreadsheets and add and delete entire graveyards from the database at a time. Also, you cannot modify existing data, such as removing a veteran in a cemetery, without deleting that entire cemetery table. This is the biggest flaw in the database design that needed to be addressed. Finally, our sponsor requested us to move the database to a private server.

3.2. Requirements

1. The application, created in Unity, must scale correctly in iOS and Android devices to be functional for iOS and Android phones as well as for iPads and Tablets.
2. We must ensure that GPS accuracy is great enough that one grave cannot be confused for an adjacent grave.
3. We must create new cemetery packs that hold information for specific cemeteries.
4. The application must be able to download individual cemetery packs.
5. The GPS must be able to locate a grave without the use of the internet.
6. The database, which runs with MySQL, must be updated so that one can add entire spreadsheets to the database at a time, rather than just one entry at a time.
7. The OCR scanning needs to be able to read text that isn't straight (i.e. curved text) and produce a one-to-one result with a veteran's entry in the database.
8. Functionality of the previous project must be preserved:
    - Augmented Reality must still point an arrow from the user to the desired gravestone.
    - GPS must still be as accurate as it was previously.
    - The database must not lose any information in the changeover.
    - The website should still be able to issue security keys, allowing access only to a select few administrators and keeping tabs on who's allowed in the system.
9. The website must keep the functionality it had before its overhaul, namely:
    a. It must be secure and have contingencies against potential attackers.
    b. It must be able to have login information stored securely on the website.
    c. It must be aesthetically pleasing and use CSS styled sheets.
    d. It must be able to track what cemeteries have been uploaded in the database.
    e. It must be able to communicate with the database.
    f. It must be able to remove cemeteries from the database as well as single entries from the cemeteries.
    g. It must be able to show who has access to the site.

10. The website must be moved off of Heroku and onto a private UCF development server.
11. The website must be improved so that it can delete entire cemeteries at a time.
12. The website must also be improved so that it can import entire cemeteries at a time.
13. The website should be able to know which graveyard packs have been downloaded most frequently.
14. The website must be able to remove and modify data of a cemetery without having to recreate the entire cemetery.

3.3 Potential Solutions

Sean Grimes

One of the issues with the previous version of this project is that it cannot be used at a graveyard while disconnected from the internet. This is because the application needs to access the database where the information of each gravestone is stored. A solution to this problem would be to store some of the data from the database locally onto the user's phone. If we were to go with this solution, we would also need to decide how much information should be stored locally. We could potentially go with storing specific gravestones data, specific graveyards data, or multiple graveyards data. Because people would likely only visit one graveyard at a time, it would probably make the most sense to separate the data into downloadable packs by the graveyard.

Tevfik Koyun

I am focusing on the website version of the project. We already have a website from the past semester however it is not very user-friendly.We will update this website to make it user friendly. This website is intended to be used by administrative users only, to update and maintain a database containing the information on the soldiers as well as the GPS coordinates of the gravesites and the individual headstones. Our plan is to use MySQL to maintain the database.
I am focusing on the back-end in this project, which includes the database and the administrative website. We've created our database by using phpMyAdmin, which is hosted by UCF's development servers. Our database is mostly inspired

from the National Veteran's Association website, which holds information on veterans and national cemeteries. We added more tables to our database as our project needed.

We want to include OCR and navigation technologies in our project. For the navigation part of the project, we have three tables to maintain those technologies and keep them separate. Also, even without the internet, the database should be able to allow users to look up information on the veterans and also provide navigation to each individual headstone in the cemetery. For these reasons, the database is one of the most important key features in our project; nothing can happen correctly without the database being securely in place. I will continue to add or delete structures within the database to meet all of the website and mobile app requirements.

I also need to work through understanding the API and its documentation for the databases. The previous team created many documents and parsing through them and applying them to a new server will probably be the hardest part.

Jordan Messler

The current mobile portion of CemetARy needs some tweaks. The app does not work well on tablets -- there is no scaling for different sizes. Unity has built-in support for this type of scaling through the Canvas Scaler component and is the most straightforward solution.

CemetARy's OCR software does not do well with non-standard text, such as the curved text engraved on many headstones. One possible fix for this is to straighten out the text so that the OCR can process it successfully. This has been known to work well for other OCR scanning solutions to headstones. Another solution is to utilize machine learning (or ML for short). I would expect ML to result in a robust OCR software; however, it would require more development time and have more opportunities to fail. Beyond that, it may not be worth the effort if other approaches can achieve similar results with less resources.

Kyle Wu

The GPS currently is decently accurate, but our version of Unity is a bit out of date and the new versions of Unity may have APIs that further enhance the

accuracy of the GPS. We'll look there to see if updating the application's version of Unity (and fixing all of the parts that subsequently break as a result of the update) can improve the application's accuracy. Even if we can't necessarily improve the GPS via updating the application's version, it's probably a good idea to do so anyway so as to allow for future iterations of the project to maintain relevancy.

3.4. Design Summary & Content

Figure 3.A. shown below illustrates the use cases a user will have when utilizing the CemetARy app. A user will be able to do four main actions from the home screen of CemetARy: scan a headstone, manually enter veteran data to search,

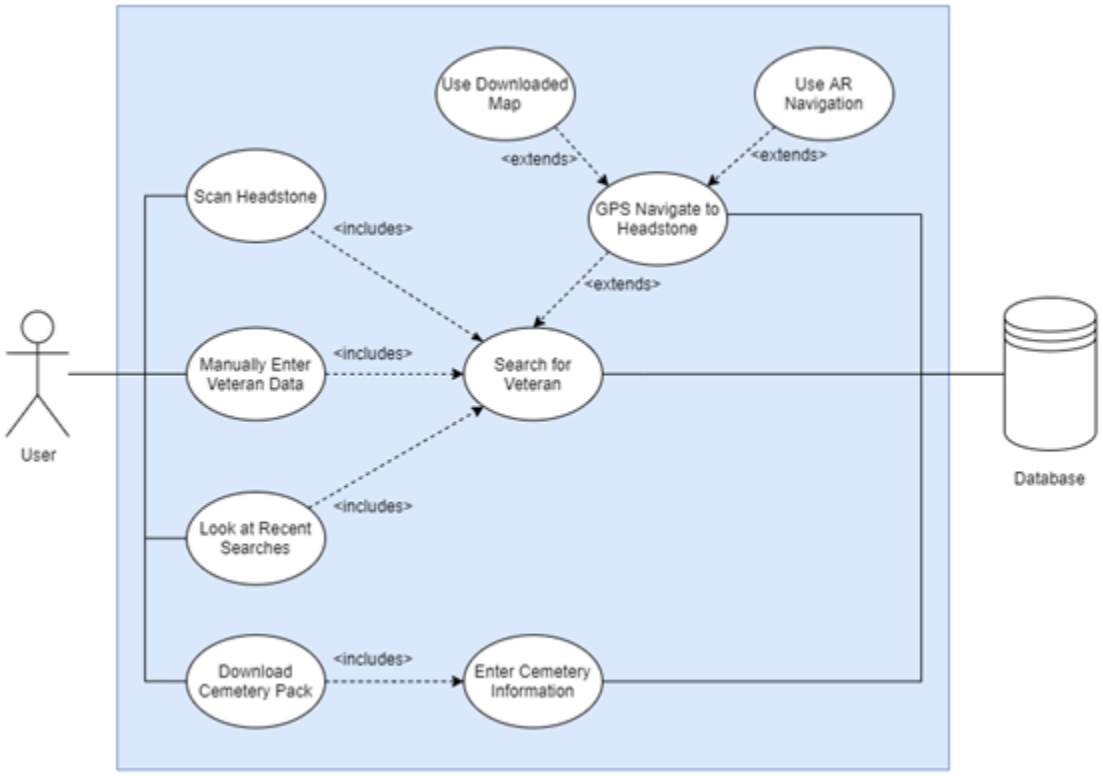


Figure 3.A. - Showcasing what a user can do with the CemetARY application

look at their recent searches, and download a cemetery pack – an offline package of a cemetery's map and data. Manually searching, scanning a headstone, and looking at the recent searches sends the information gathered to the database for a search of the veteran. Once a profile is obtained, the user can begin GPS navigation to the headstone of the veteran, which offers an offline

mode provided they downloaded a cemetery pack ahead of time. They can also swap between the map and AR modes during any point of navigation.

The CemetARy app is designed in this way so that every option a user has will lead them to the veteran profile. This is so a user can use the app like a standard database if they want to read about a veteran (or many) in the cemeteries. This would not require them to be at a cemetery. However, we expect most users of the CemetARy app to be visitors, perhaps first visitors, that do not know their way around the cemetery grounds. Being able to easily find a way to navigate to their desired location is the main goal of the app, so every veteran profile has an apparent option to start navigation to the veteran's headstone. This will hopefully make cemeteries more accessible to the general public, since many are large and uniform, making them difficult to navigate.

4. Administrative Content

4.1. Budget and Financing

The table below shows the cost of development. The sections in gray are potential future costs of maintaining CemetARy and are not factored in the current total.

Asset	**Cost**
OpenCV	$95
In-App Web Browser	$12
Unity Storage	$5
Mapbox	$4/1000 monthly users (over first 25,000)
Google Vision	$1.50/month (after first 1000 units)
Play Store Availability	$25

Apple Store Availability	$99/year
CURRENT TOTAL	**$112**

4.2. Milestones

Senior Design 1

Date	Milestone	Status
6/4/2021	**Initial Meeting with Dr. Giroux**	**Completed**
6/11/2021	**Second Meeting with Dr. Giroux**	**Completed**
6/13/2021	**Initial Design Document Due**	**Completed**
6/18/2021	**Develop mock-up for new website layout**	**Completed**
6/18/2021	**Third Meeting with Dr. Giroux**	**Completed**
7/1/2021	**Meeting with OCR Team**	**Completed**
7/2/2021	**Complete research on Unity Update**	**Completed**
7/7/2021	**Construct Dummy Database**	**Completed**
7/9/2021	**Fourth Meeting with Dr. Giroux**	**Completed**
7/11/2021	**Second Meeting with OCR Team**	**Completed**
7/13/2021	**In-Class Presentation**	**Completed**
7/14/2021	**50% Design Document Due (60/120 Pages)**	**Completed**
7/16/2021	**Connect Server to Dummy Database**	**Completed**
7/23/2021	**Fifth Meeting with Dr. Giroux**	**Completed**

8/5/2021	**Final Design Document Due**	**Completed**
8/6/2021	**Sixth Meeting with Dr. Giroux**	**Completed**
8/6/2021	**Summer Semester Ends**	**Completed**

Senior Design 2

8/23/2021	**Fall Semester Begins**	**Completed**
8/27/2021	**Connect Website to Backend**	**Completed**
8/27/2021	**Seventh Meeting with Dr. Giroux**	**Completed**
9/1/2021	**Administrative Website Functional**	**Completed**
9/10/2021	**OCR Improvements Functional**	**Completed**
9/10/2021	**Functional Website Completed**	**Completed**
9/10/2021	**Eight Meeting with Dr. Giroux**	**Completed**
9/24/2021	**GPS Improvements Functional**	**Completed**
9/24/2021	**Ninth Meeting with Dr. Giroux**	**Completed**
9/30/2021	**Connect Application to Backend**	**Completed**
10/8/2021	**Tenth Meeting with Dr. Giroux**	**Completed**
10/10/2021	**OCR Improvements Finalized**	**Completed**
10/20/2021	**Website Completed**	**Completed**
10/22/2021	**Eleventh Meeting with Dr. Giroux**	**Completed**
11/1/2021	**GPS Navigation Completed**	**Completed**
11/5/2021	**Twelfth Meeting with Dr. Giroux**	**Completed**

11/15/2021	**Final Application Completed**	**Completed**
11/30/2021	**Final Testing Completed**	**Completed**
12/1/2021	**Final Project Presentation & Evaluation**	**Completed**
12/3/2021	**Final Design Document Due**	**Completed**

4.3. Project Planning

We planned our project and resource management using the Gantt chart in Figure 4.A. There are 3 sections: Research (the plan towards learning materials), Development (the true progress on the project), and Documentation (the design documents and code documentation). Some nice additional features used were checklists for each objective, and percentage progress based off those tasks for an objective. This provided a strong gauge of where we were in the project. Example images of the checklist and the percentage progress can be seen in Figures 4.B. and 4.C.

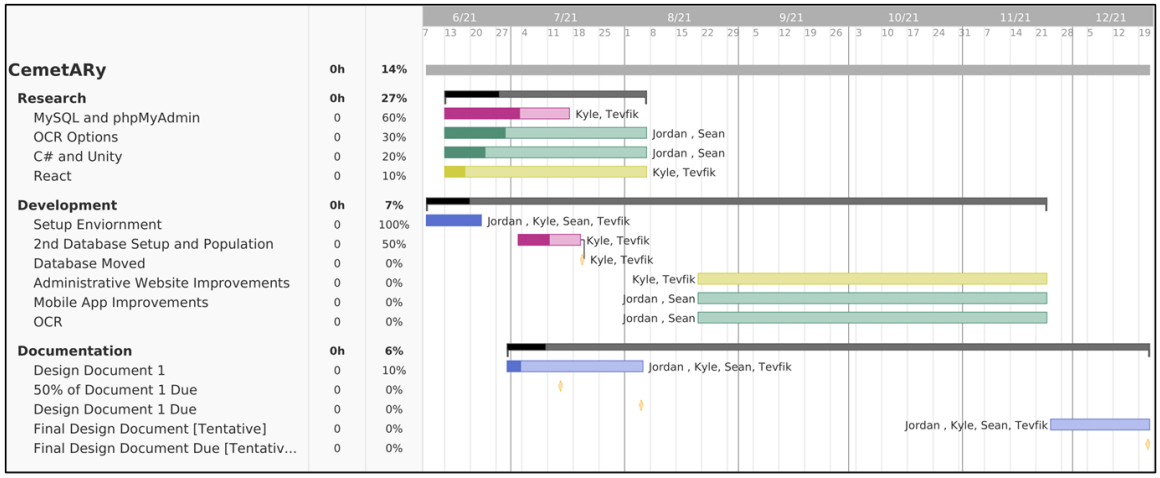


Figure 4.A. - Our team's Gantt Chart

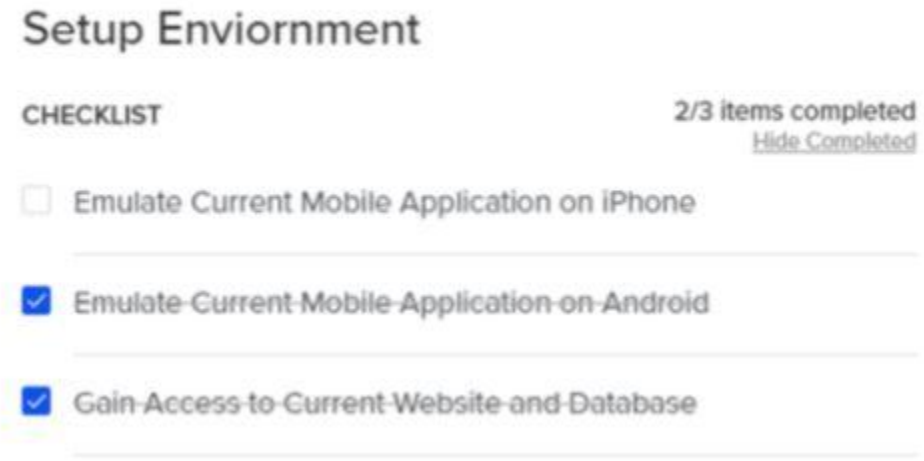


Figure 4.B. - Our team's Gantt Chart checklist

Figure 4.C. - Our team's progress in the Gantt Chart

For more active updates and flexible work, the entire team used a task board through Trello (Figure 4.D. below). This made it easier to see what each team member was working on and was managed by all four members unlike the Gantt chart. While the Gantt chart is nice for overall progress, the task board made it much easier to visualize what main main goals of the project were done, what was being worked on, and what we still needed to start (as seen by the "To Do," "Doing," and "Done" columns. This way, we could change our workflow and resources in correspondence to what tasks were being worked on.

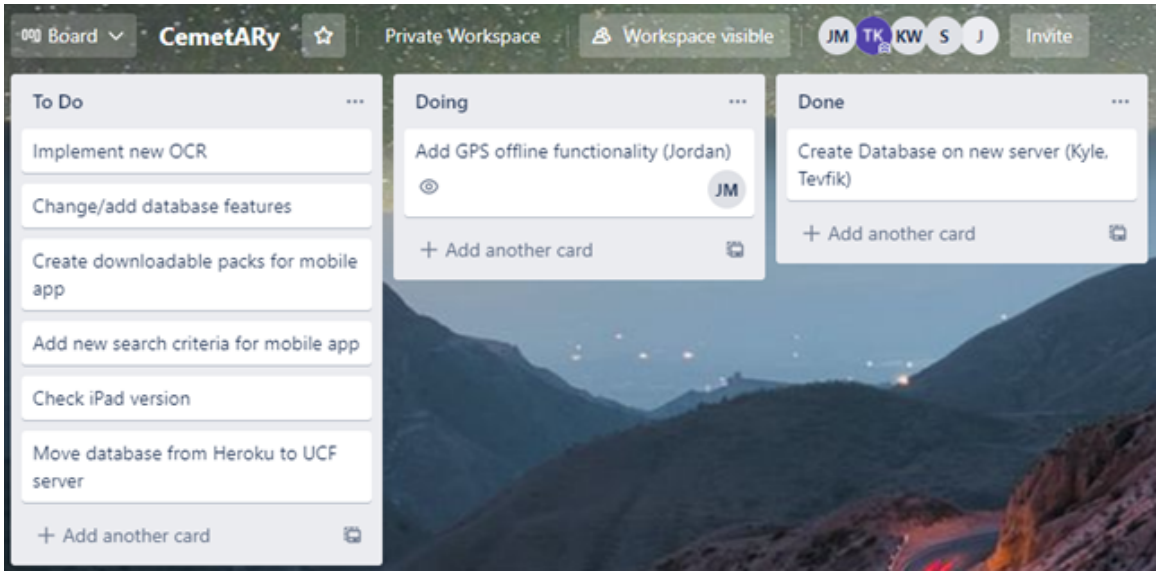


Figure 4.D. - Our work displayed on Trello.

4.4. Project Block Document

As can be seen from the project block diagram in figure 4.E., there are several components to this project. The two biggest components are the website and the mobile application. These two components are the components that other users are going to be using. Users visiting graveyards will be using the mobile application, and administrators changing the database will be using the website.

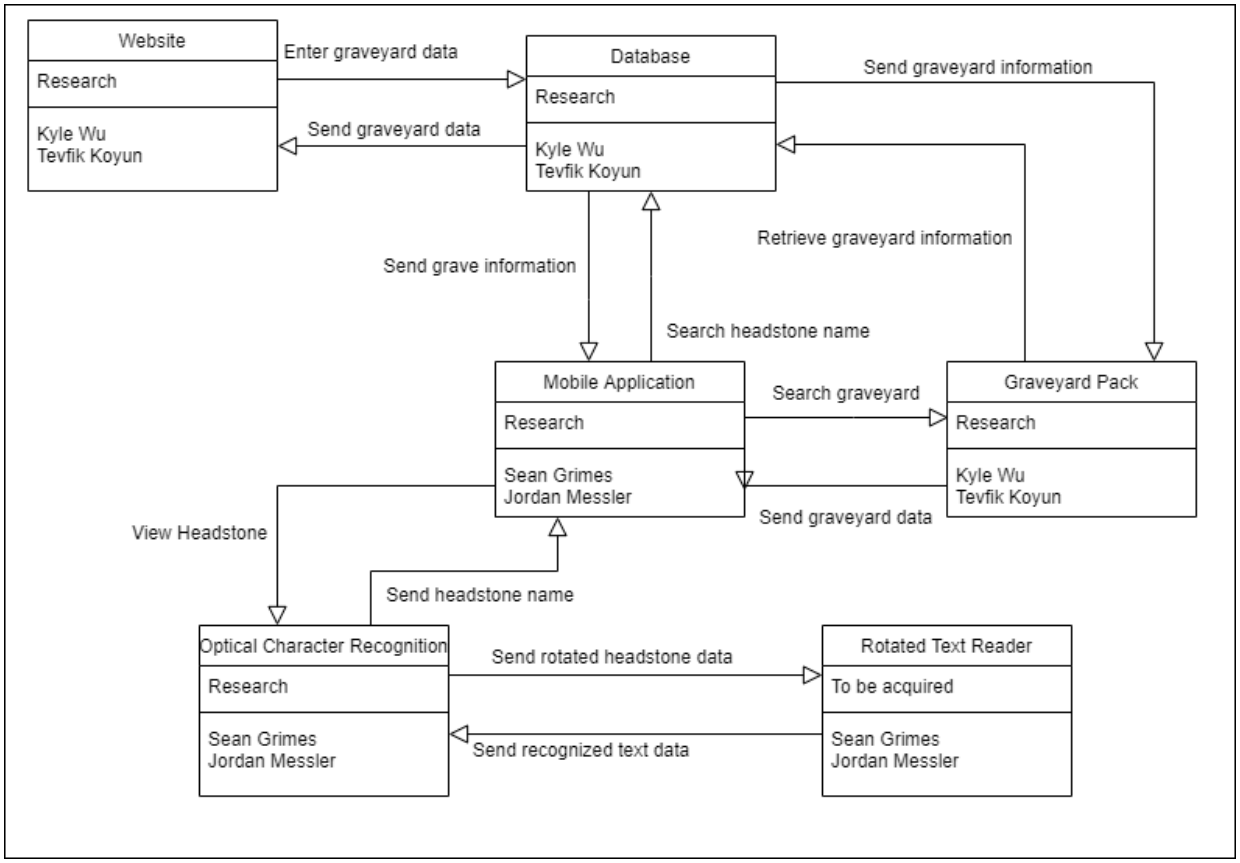


Figure 4.E. - Our Project Block Document

We decided to divide responsibility of this project into two groups, where one group is working mainly on the website and database, and the other group is working on the mobile application. Kyle Wu and Tevfik Koyun worked on the website and database, and Sean Grimes and Jordan Messler worked on the mobile application.

5. Research

5.1. Database

5.1.1. Database Requirements Reiterated

1. The database server needs to be transferred to the UCF server.
2. The mobile application should be able to reach the location information on the database.
3. The APIs from the mobile application should be able to search tables from the database, including tags such as name, cemetery, section.
4. The previous team had an API for encryption, but it is slow and we need a better solution.
5. The sponsor may have additional requirements that need to be satisfied as mentioned.

5.1.2. MySQL Requirement

In the previous iteration of the project, Dr. Giroux had the team use MySQL as the database for the project for several reasons. Firstly, Dr. Giroux is very familiar with MySQL and so can offer both new and older members of the project her expertise on the subject. Furthermore, having MySQL as a database allows for an easy hand-off between project members because it is a very commonly used database language.

Finally, the schema for MySQL existed previously for the Veterans Legacy Program, and had been given to the previous team as well. This matching schema allowed the previous team very high fidelity to the existing program's database.

We have maintained the same database structure and so have decided to continue using MySQL over other databases such as MongoDB, which would require an overhaul of the entire system.

5.1.3. Stored Data

The previous team decided to use MySQL for all of their database and information logging needs, and as a result, our team will continue to work with the MySQL language. The database we're working with is supposed to keep all of the information on the veterans, cemeteries, and sections in the cemeteries that the veterans are buried within.

Additionally, the database holds the exact longitudinal and latitudinal points for each of these veterans, as well as other information, such as the dates these veterans were born and died, their ethnicity, the wars in which they fought, and their gender. These data provide the mobile application of this project both navigational features as well as search functions.

The previous team did not make any attempts to collect data from application users; our database might very well not as well. It will be a stretch goal if we've completed the other tasks in our requirements to perhaps allocate some time and resources to developing a way of tracking user activity on the mobile application and saving the data to our database as well. Finally, the mobile application needs to run offline as well, and to that end, we will be implementing cemetery packs that can be downloaded separately and allow the user some limited access to our databases in that regard.

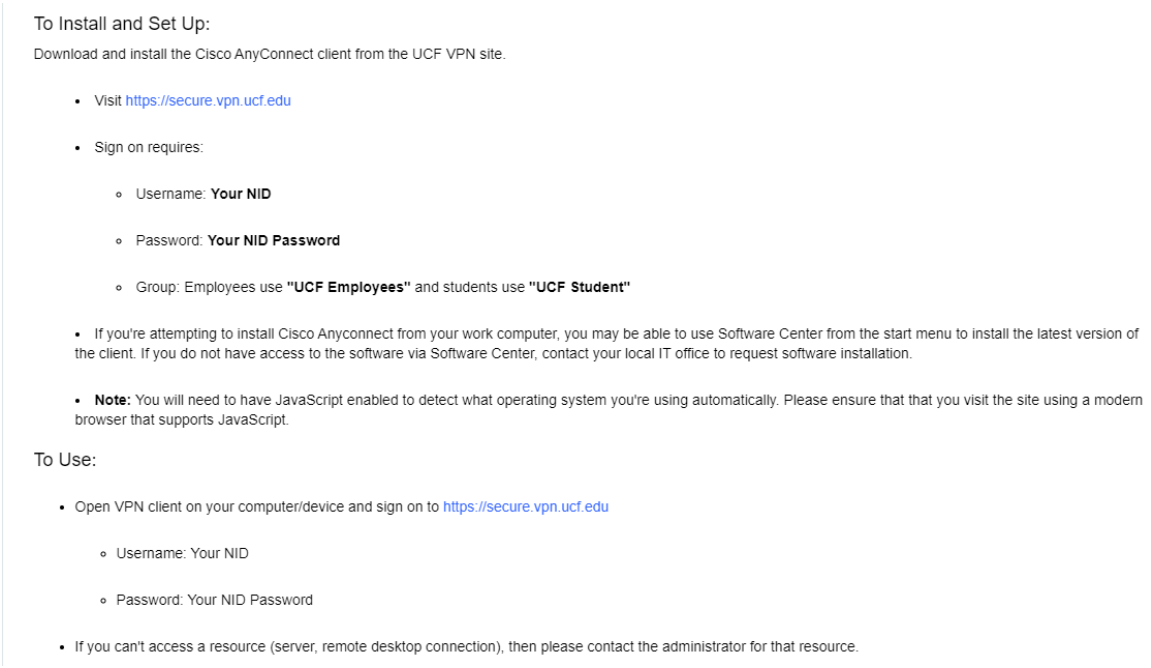
5.1.4. - Hosting the Database

The previous team used Heroku to host the database, among many other options such as AWS, and Google Cloud. However, for our project, Dr. Giroux wanted to use UCF's private development server instead. The previous team had tried to implement this server switch in the previous iteration of the project, but weren't able to due to mechanical constraints at the time.

There are some security benefits to having the database on UCF's development server, but there are also some drawbacks as well. Work on the development center can be halted a little bit if precautions are not made to be able to readily access UCF's VPN.

Since UCF is a private server, in order to use and build the database, we will need to have access to UCF's VPN. In order to do that, we need to follow UCF IT Support's guide, which will be explained in the subsequent sections.

The UCF IT Support has a very convenient and easily searchable downloadable VPN. In order to use the VPN, the user is prompted to download and install the CiscoAnyConnect Client™.



To Install and Set Up:

Download and install the Cisco AnyConnect client from the UCF VPN site.

- Visit https://secure.vpn.ucf.edu
- Sign on requires:
  - Username: **Your NID**
  - Password: **Your NID Password**
  - Group: Employees use **"UCF Employees"** and students use **"UCF Student"**
- If you're attempting to install Cisco Anyconnect from your work computer, you may be able to use Software Center from the start menu to install the latest version of the client. If you do not have access to the software via Software Center, contact your local IT office to request software installation.
- **Note:** You will need to have JavaScript enabled to detect what operating system you're using automatically. Please ensure that that you visit the site using a modern browser that supports JavaScript.

To Use:

- Open VPN client on your computer/device and sign on to https://secure.vpn.ucf.edu
  - Username: Your NID
  - Password: Your NID Password
- If you can't access a resource (server, remote desktop connection), then please contact the administrator for that resource.

Figure 5.A. The UCF IT Support provides a comprehensive guide to their VPN

After completely downloading the Cisco application, opening the application then brings up the dialog box in the following figures. These dialogue boxes will allow UCF students and faculty to sign in.

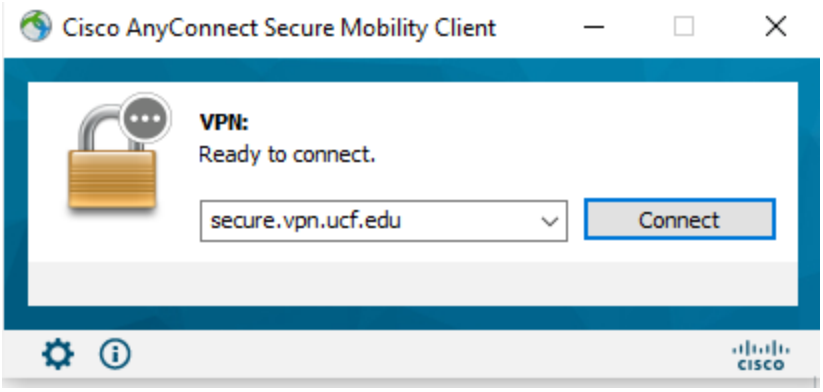


Figure 5.B. This is the CiscoAnyConnect Client™ opening screen.

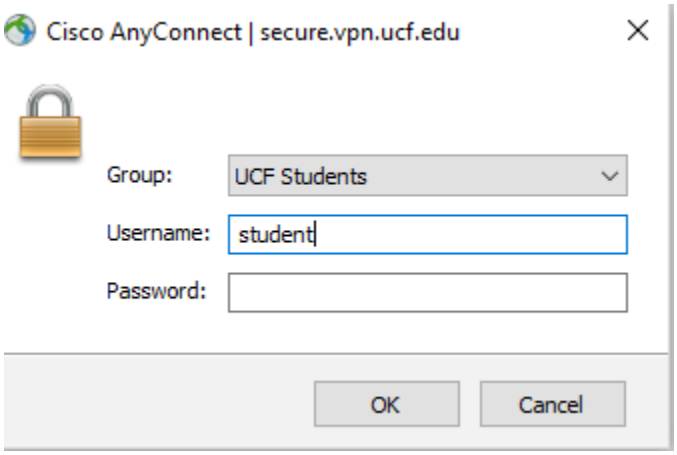


Figure 5.C. This is the CiscoAnyConnect Client™ login section.

After connecting to the VPN Server, one can access the private UCF server that hosts the database by accessing the following URL:

http://chdr.cs.ucf.edu/phpmyadmin/

Only users who have been granted access by the sponsor can enter the database.

Figure 5.D. PhpMyAdmin login page through UCF server

Once login information has been granted to the user, he or she can access the database and see and edit its overarching structure.

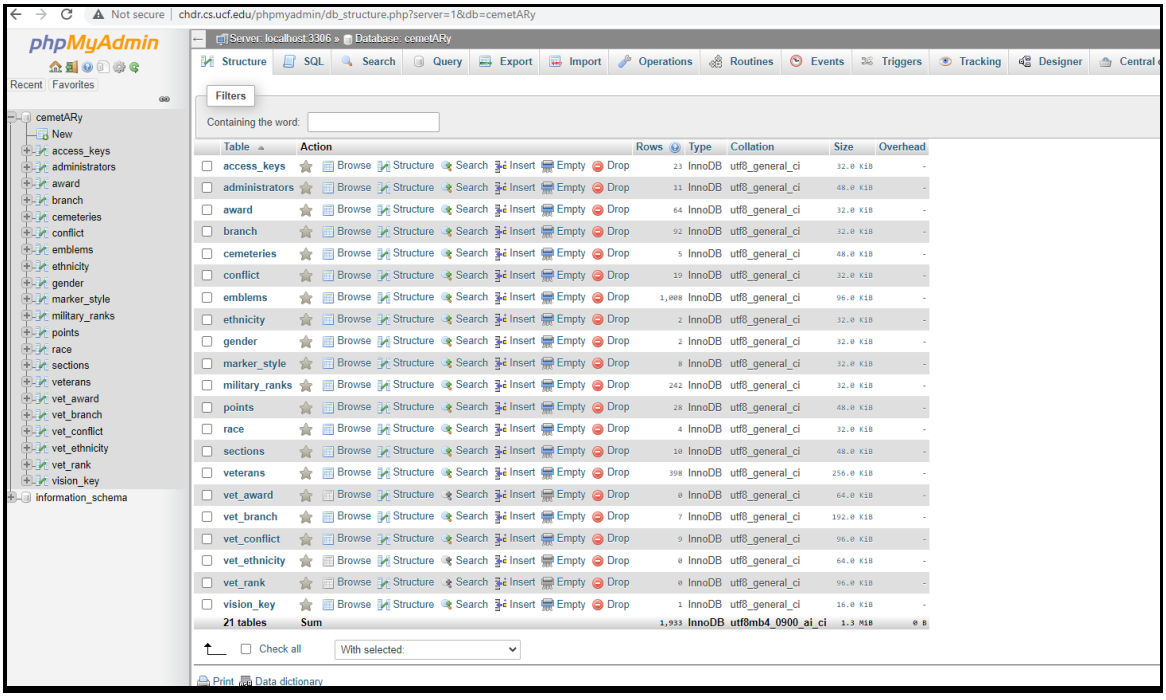


Figure 5.E. PhpMyAdmin Database tables

There were many benefits to switching the site over from Heroku to phpMyAdmin. First and foremost, Heroku had an upkeep cost associated with it.

By contrast, PhpMyAdmin is a free software tool written in PHP intended to handle the administration of MySQL over the Web. Also it supports foreign keys and InnoDB tables

PhpMyAdmin is very simple to set up as a tool and provides user-friendly graphical interfaces to access website related data. PhpMyAdmin has a designer section.This section helps to show each table relationship like Figure 2.2 .Since the project needs to be able to import and export on demand, choosing phpMyAdmin makes perfect sense; phpMyAdmin can import and export data in various formats like .CSV, .XML, .PDF, Word, Excel and other such extensions.

5.1.5. Database Structure

The previous team created many tables in order to hold all of the information on the veterans in every graveyard. The design of these tables closely mirrored the structure that the veterans Legacy Project had because the mobile application relies on data drawn from that project. Besides the tables related to the cemeteries and the veterans, there is also a table dedicated to holding information on administrator users and for website maintainers.

The Figure 5.E. that follows visually explains the schema that the National Veterans Association currently uses for the veteran information they have collected. Dr. Giroux provided this scheme to the previous group as well as to our group. At first glance, the schema is not the easiest to parse. However, the table can be fairly well understood when explained.

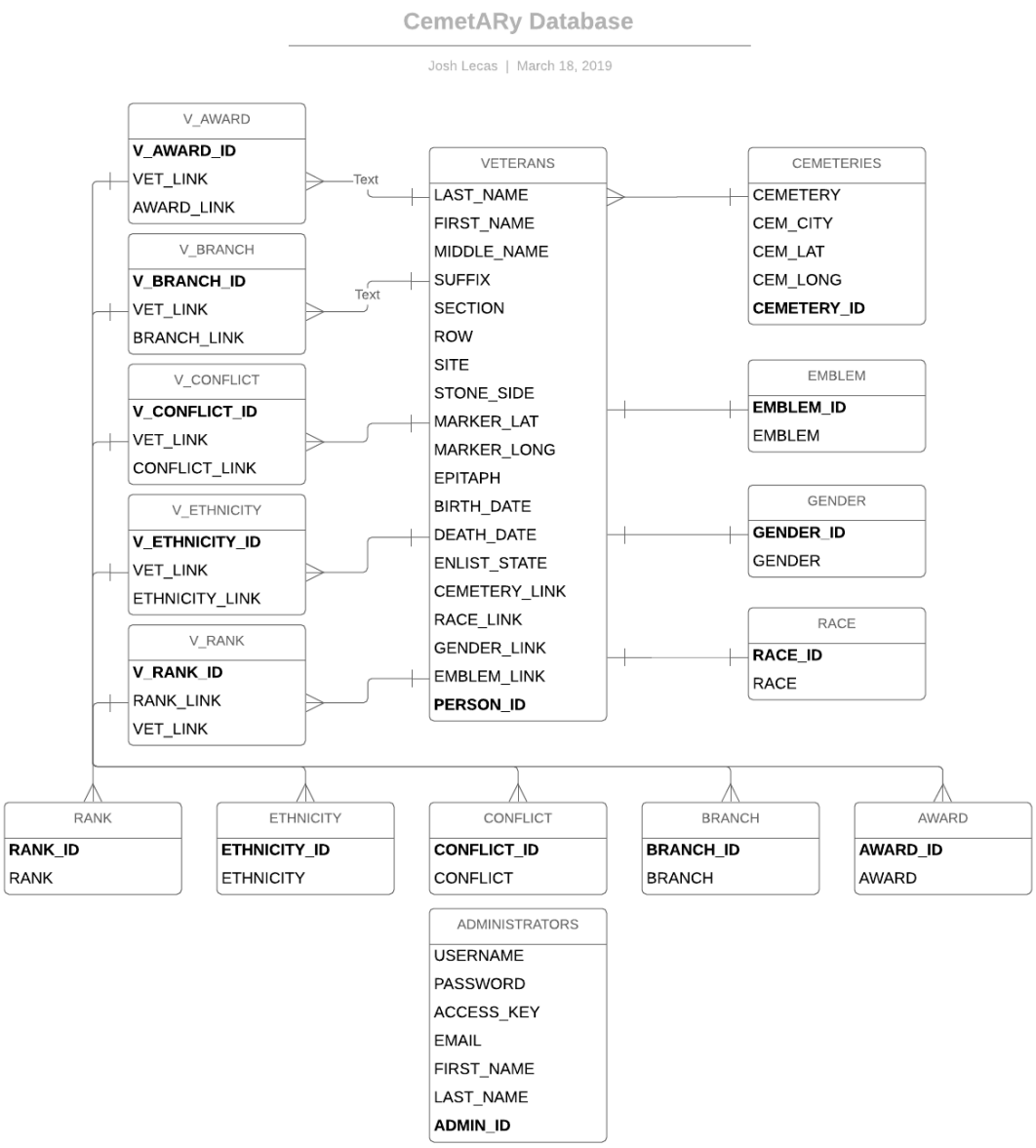


Figure 5.F. Initial Database Schema provided by Dr. Giroux

In this schema, the main table is the VETERANS table, which most of the data collected is sent to. In order to expedite search functions and other practical applications of the database, the VETERANS table has relationships with other smaller tables. These relationships allow one to search through the database without having to call the VETERANS table for each individual search. Also there are some tables, namely, the ADMINISTRATORS that have no relations with the main table.

The schema has four tables on the right side. These tables are the Marker Style ,RACE, EMBLEM, CEMETERIES, and GENDER tables. These tables are considered child tables for the veterans table and they each have one to one relationship with the VETERANS table. The goal of these tables is to help the administrators maintaining the database to easily add to the VETERANS table without duplicating the data.

The left side of the schema contains five tables. These tables are RANK, ETHNICITY, BRANCH, CONFLICT and AWARD tables. These five tables have many to many relationships with the VETERANS table and individually each of these five tables has a one-to-one relationship with the v_ version of itself. The previous team called these v_versions ‘auxiliary tables’.

The purpose of these v_versions is that there could be cases of veterans serving in more than one conflict, being in more than one branch of the military, or having more than one award within their lifetime. In actuality, the chance that someone has many different types of data pertaining to them is very high. In order to preserve the many-to-many relationship in the database, each v_version table needs to hold the primary key pair for the other version of itself. By using these pairs of primary keys, the database and API will be able to connect the tables together.

While at first glance, in the many-to-many relationship tables, ETHNICITY and RANK tables look like they should have a one-to-one relationship with each veteran, the opposite could not be more true. These tables cannot be implementable as a single column.

Dr. Giroux has informed us that the Veterans Legacy Project has noted that many of the veterans buried within the national cemeteries have hailed from different countries and have a mix of ethnicities. In terms of RANK, in the event that a veteran has served in more than one branch of the military, he or she will likely have different ranks within the different branches. As a result, ETHNICITY and RANK tables, like the BRANCH, CONFLICT, and AWARD tables, must keep their many-to-many relationships to the VETERANS table.

5.1.6. Additionally Required Tables

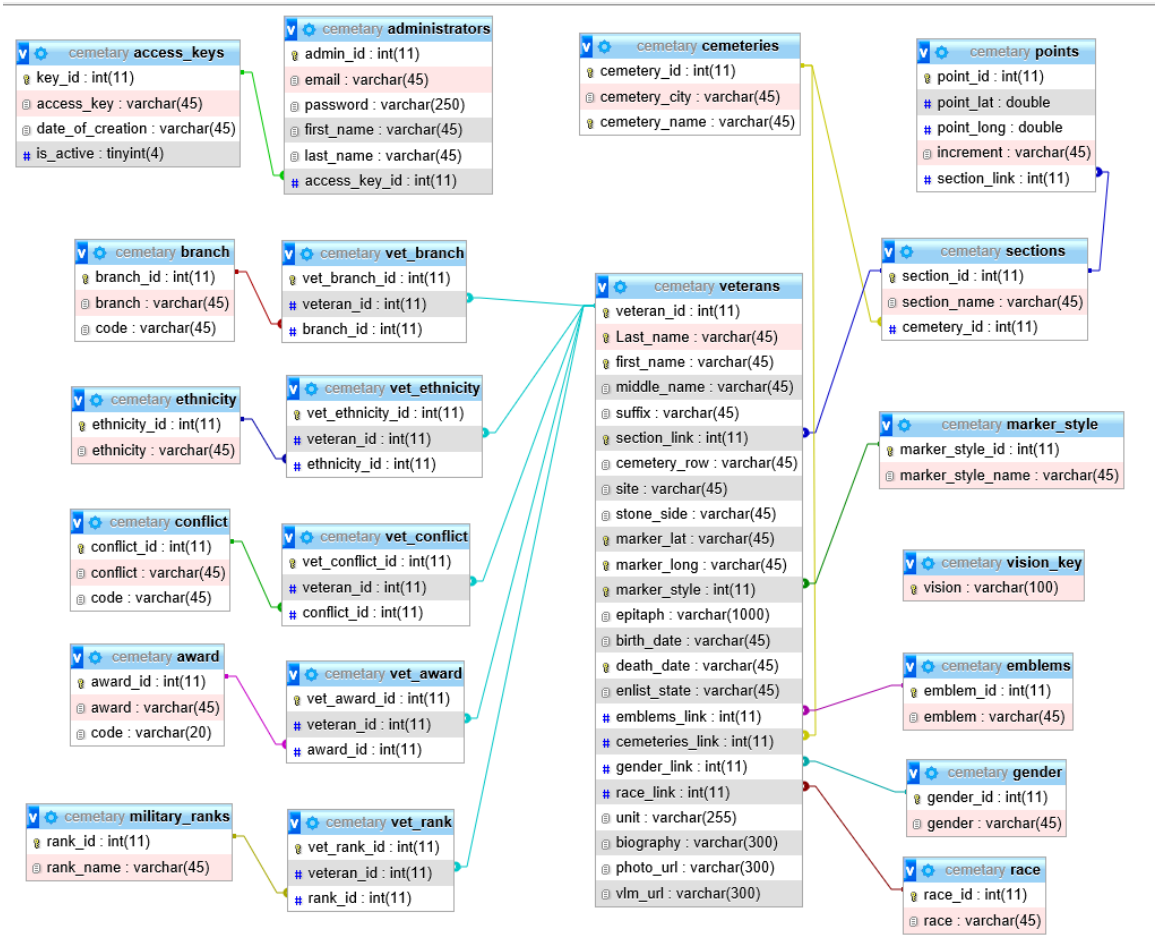


Figure 5.G. Modifications have been made to the initial schema.

The previous team noticed that besides the original schema, they needed to add the ADMINISTRATORS, ACCESS_KEYS, SECTIONS, and POINTS table. Figure 2.2 shows the current version of the database, which still closely follows the previous team's schemas.

There are two tables which have no relationship with the VETERANS table or any other tables. These tables are the ADMINISTRATORS table and the ACCESS_KEYS table. These tables have a one-to-one relationship with each other. The ACCESS_KEYS table has both active and inactive keys that are used to help new administrators sign into the website that will control the database. Additionally, holding the information for the access keys gives the administrative

website the ability to not only generate new keys, but also find what keys have been created and to whom they are allocated.

The ACCESS_KEYS table holds a primary key which is the KEY_ID. With that KEY_ID, the ADMINISTRATORS table can keep track of the access keys on the website. Furthermore, the ADMINISTRATORS table holds the first and last names of the administrators of the website, as well as their email, password, and ID. The ADMINISTRATORS table will not be accessed by the mobile application at all; it is only constructed to help maintain and secure the administrative website.

The previous team added a SECTIONS and POINTS table in order to further improve the mobile application. Both of these tables help narrow down the location of each veteran's headstone. Specifically, the national cemeteries are divided into different sections, and each section has a set of geological data points that make up the corners of the section. The CEMETERY tables have a one-to-many relationship with the SECTIONS table, while the SECTIONS table has a one-to-one relationship with the POINTS table.

The mobile application will use the SECTIONS table in order to help estimate where the desired veteran's headstone is located. Once the user of the application has reached the correct section of the cemetery, the application will then narrow down the navigation to the desired headstone. The navigation portion of the app will use these sections as an estimate of where a headstone is located to help users get navigated to a section, and then the app will refine the navigation down to the headstone itself.

These new additions to the database help the other components of the project, namely the administrative website and the mobile application function as per the requirements.

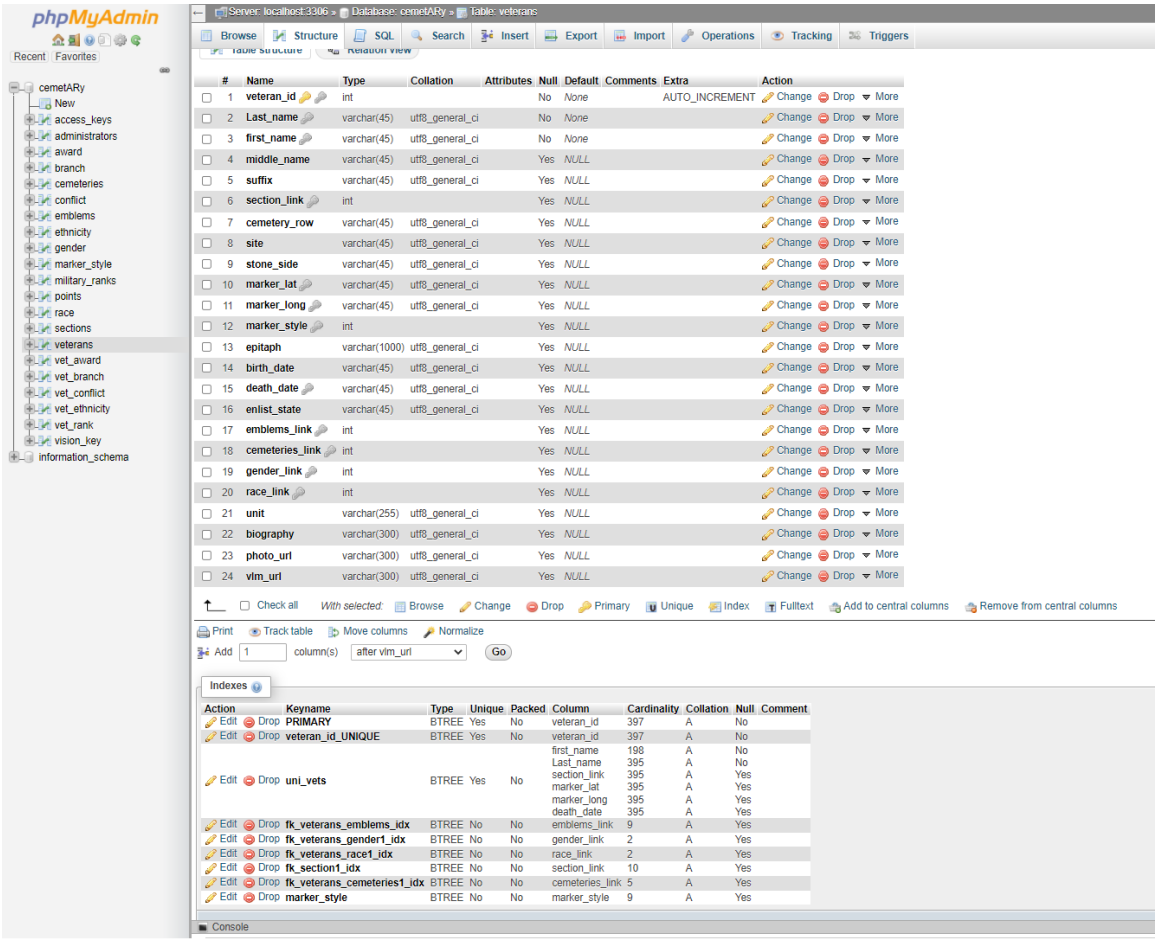


Figure 5.H. - VETERANS table from database

As previously stated, the VETERANS table is the main table in this database. There are many other tables that have identical data to the VETERANS table. To protect the main table from duplicated data, the VETERANS table has a unique index which prevents exact duplicates of veterans from being added to the table.

Dr. Giroux additionally wants us to add two URL columns to the veteran table: the photograph URL and VLM (Veterans Legacy Memorial) URL. The photograph URL will point to the Veterans Legacy Project web server, which will show up on the mobile application when a veteran is searched. The VLM URL will show information pertaining to the veteran's biography if one exists.

5.1.7. API Endpoints

API Endpoints allow both web and mobile applications to communicate with our database. In this semester, we are trying to understand how the previous team used API endpoints to connect their application and their website to their database. These end points might be changed in the future when we develop the database further and implement new database features such as mobile data entry to our database.

The previous team used the Express Middleware function for routing. Middleware functions are useful in that they can separate API definitions into different parts. Since our project has a website portion and the mobile app, the portion will require different accesses and requests from the database. Using these Middleware functions will be much easier for us than programming them from scratch.

The previous team decided to divide up the three major sections into administration, data retrieval, and data entry. Administration is related to the database's Administration table, which controls the access keys and administrator information pertaining to the website. Data retrieval pertains to making endpoints to access the database from the mobile application. Finally, data entry pertains to adding and deleting information from the database, again primarily from the website.

These portions may need to change if our requirements need us to log data from the mobile application, but at the present time, we will continue to use the previous team's approach. The current API is written in Javascript using the Node.js and Express.js server frameworks.

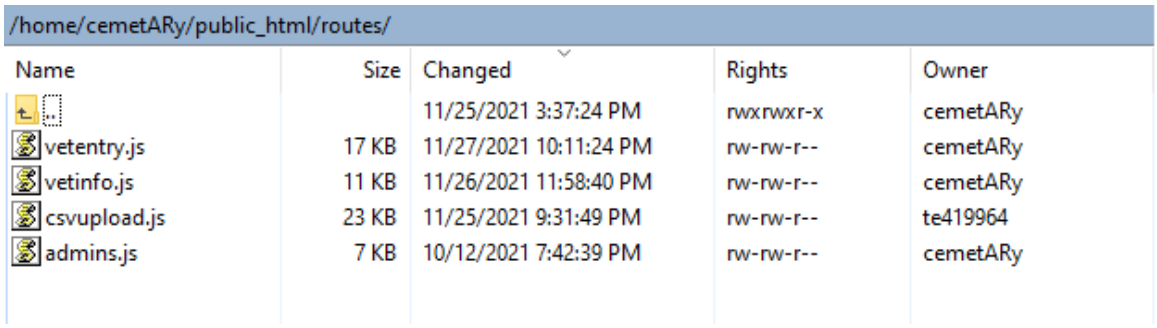


Figure 5.I. - A list of the used API

As previously mentioned,In order to maintain simplicity and modularity of the project, the API Endpoints are divided into three groups, which are: admins.js, vetentry.js, and vetinfo.js. visually displayed in the figure above.Below is the list of files and explanations for each of their API Endpoints.

ADMINS.JS

The ADMINS.JS endpoints require the use of JSON Web Tokens, which, alongside the login endpoint, will provide the user with the appropriate token. Once the user has correctly input their login information, these endpoints will allow maintainers to access the administration table on the database.

Register - POST

The Register endpoint is used for the access key table on the database and allows new users registration.This endpoint is required for the website portion of the project in order to connect the website and the database. The register endpoint requires the website to send the registration data from access key contents.

The Register endpoint will look at the access keys and compare from the table and check if the key is currently active or not. If the access key is not active, then the user's chosen password will be encrypted and saved to the database. After that, a new admin will be created. The previous group has decided that whenever an admin is created, the endpoint makes another query to update one of the inactive keys to change to active.

In the event that one of the processes returns an active access key or creates an admin that already exists, the Register endpoint will return an error code.

Login - POST

The login endpoint will check the username and the password as well as if the user exists and the password matches. If the user is present within the system and the input password is correct, the login endpoint will generate a JWT token authentication. After that, the website will continue to the home webpage. The

Login endpoint receives the username and password from the JSON payload. After receiving the information, the password will be decrypted.

GenerateKey - POST

The GenerateKey endpoint will generate a specified length of random keys.
These keys will be added to the database to the table of keys. These new added keys automatically will become inactive keys.

List Keys - POST

The List Keys endpoint is used for the maintenance of the website portion of the project. This endpoint allows the website to display each key's status for either distribution purposes or checking which keys are active. The List Keys endpoint is related to two other endpoints: the Register and GenerateKey endpoints.

The List Keys endpoint receives the active keys and administrator information from the Register endpoint, as well as the inactive keys from the GenerateKey endpoint.

VETENTRY.JS

The VetEntry.JS holds endpoints that need a token along with a request in order to be processed. The user receives these tokens from the Login endpoint.

NewVet - POST

The NewVet endpoints are used for the dynamic creation of a veteran object. These endpoints also allow the user to enter veteran information into multiple tables at once and are dependent on information coming in from the JSON Web tokens.

For the veterans table, NewVet endpoints create as many empty arrays as required after receiving JSON Web tokens. After that, when the arrays get filled with data from the web tokens, these data will be used for building the SQL query. That query checks whether all the fields which are arrays are filled with data or not.

After all the fields have been checked, the data will be sent out by SQL query to several other fields such as conflict, ethnicity, branch, etc. However these tables contain many-to-many relationships with each other. If we send out the data to all of these fields at once, there may be inconsistencies in the data because we're updating multiple fields at the same time.

For this reason, each table should be filled up with separate queries. Lastly, the system will check whether the veteran's table has been completely populated. If all has gone correctly, these tables should be filled up. If not, the process will cancel itself and display an error. This condition of requiring a fully populated veteran's table exists in order to prevent errors such as duplicating data.

NewCemetery - POST

The newCemetery endpoints will not often be used in this project. The NewCemetery endpoint starts after all API endpoints and projects are done. This endpoint will be used when the user tries to add a new cemetery to the database. Additionally the existence of the NewCemetery endpoint makes it easier to add to the database. This endpoint receives data from the JSON payload and adds them to the database. Examples of data received are endpoints like the NewVet endpoints and the NewSection endpoints.

NewSection - POST

The NewSection endpoint is very similar to newCemetery endpoints. The difference between these endpoints is that the NewSection has ID columns. This endpoint allows the user to search through cemeteries by sections.

NewPoint - POST

The NewPoint endpoint is similar to the NewSection and NewCemetery endpoints. The difference in the NewPoint endpoint is that it not only receives an ID for the section for which it is associated, but it also obtains the point's latitude and longitude, as well as an increment variable. This endpoint is not currently used in this project. However for future development, it will allow the user to add more specific search functions related to locations.

Deletion - POST

The Deletion Endpoints are related to all endpoints located within the VETENTRY.JS, such as the NewVet, NewCemetery, NewSection and NewPoint endpoints. These endpoints receive object IDs from the database. After receiving the ID, the user is allowed to delete objects from the database.

Edit - POST

The Edit Endpoints receive from the website on the Veteran Info page changes to the veterans table that need to be uploaded into the database. After receiving the changed values, it then returns the values back to the website so that the table being displayed in the website gets updated as well.

VETINFO.JS

The VETINFO.JS holds the endpoints that do not need a token along with a request for them to be processed as JSON tokens. These endpoints will only receive information from the tables and do not directly interact or manipulate the database.

SearchForVet - POST

The SearchForVet endpoints are the most commonly used endpoints in this project. These endpoints create a dynamic SQL query. Creating a new query will depend on the filled-out fields which will be sent to the server via JSON request. This request uses the SQL keyword in order to check only partial sections such as admin_name. This partial checking allows for much more accurate searching and this accuracy will help prevent potential errors.

The SearchForVet endpoint adds new sections of SQL query to a string to be used. The string used depends on the field included in the JSON request. This new section creates new relationships in the database: i.e. the connection between the CONFLICT table to the VETERANS table.

ViewAllVets - GET

The ViewAllVets endpoint simply receives information from the Veterans table in the database. After obtaining the information, the endpoint simply returns a list of data about Veterans to the user. These endpoints are needed for the veteran_full view table. This view table is just for the website portion of the project.

However, this endpoint also requires additional IDs on the veterans table related to the cemetery and sections table. To complete the list all about the Veterans, the information pertaining to both the cemetery and sections tables need to be added to that list. Without the information on both the cemetery and sections tables, the endpoint cannot be used properly. This endpoint is particularly helpful for the website maintainers in that they will be able to view all of the existing veterans currently located in the database at the same time.

Cemetery Information Endpoints

Additionally, there are three more endpoints related to receiving data about the Cemetery. These endpoints are the ViewAllCemeteries, ViewCemeterySections, and GetSectionPoints, and they are used to search data on the website portion of the project.

ViewAllCemeteries

The ViewAllCemeteries endpoint collects data from the cemetery_table and returns a list of tables pertaining to the cemetery table. Entities included on that list are tables such as cemetery_ID, cemetery_name and cemetery_city.

ViewCemeterySections-post

The ViewCemeterySection endpoints is a post. The post operates in a similar way to the ViewAllCemeteries endpoint. The post collects the data from the section tables and returns the section_ID to the user.

GetSectionPoints

The GetSectionPoints endpoints also function in a similar way to the ViewAllCemeteries endpoint. However with this endpoint, the section ID is required in order to obtain the section points. Without first receiving the section

points, the database's foreign keys will not allow the GetSectionPoints to continue. Essentially, the endpoint first needs to receive a section ID and then it can return a section point to the user.

CSVUPLOAD.JS

CSVUpload

The last API pertaining to the cemeteries is the CSVUpload endpoint, which is large enough to be contained within its own JS file. The endpoint is heavily detailed within its own JS file, but an additional explanation will be given below.

The goal of the CSVUpload endpoint is to take in data provided by both the database as well as a well-formatted .csv file and parse the .csv's contents to the database. Six functions exist within the export: (1.) the masterFunction which holds the other functions and ensures they run in the proper order, (2.) the getSection function which parses through the .csv once to find all of the sections and add them to the cemeteries prior to adding the veterans, (3.) the columnFinder function which parses through the first veteran to determine which columns are present within this particular .csv file, (4.) the asyncVetInsert function which first waits for the (5.) getVetID function to insert the veteran into the veteran table before using the ID obtained to insert the veteran's awards, branches, conflicts, and ranks into their respective tables, and finally (6.) the fileInsert function which logs any errors parsed into an errorlog.

This uploader can parse about 2500 veterans in eight minutes.

SPECIFICATIONS ON .CSV FILES

There are some differences between the .csv headers and the database's fields. Listed below are the headers required for certain fields within the .csv to be parsed correctly into the database. Note that while the veterans fields themselves need to be in all capital letters in order to match with existing records, the field headers themselves are case-insensitive, as they'll be parsed to lower-case while running through the functions. The headers within the .csv file can be in any order, and only SURNAME and FIRST_NAME are absolutely required.

CSV Header	Relevant Database Field
SURNAME	Last_name
FIRST_NAME	first_name
MIDDLE_NAME	varchar
SUFFIX	suffix
SECTION_NAME	section_link
ROW	cemetery_row
SITE	site
FACE	stone_side
LATITUDE	marker_lat
LONGITUDE	marker_long
MARKER_STYLE	marker_style
EPITAPH	epitaph
BIRTH_DATE	birth_date
DEATH_DATE	death_date
STATE	enlist_state
EMBLEM_BELIEF	emblem_link
SEX	gender_link
RACE	race_link
UNIT	unit
BIO_URL	biography
PHOTO_NAME	photo_url
VLM_URL	vlm_url

There are additional columns that parse to other tables as well. The header RANK can be used to send a rank to the vet_rank table. Branch columns must be of the form BRN_xx to insert into the vet_branch column, where ‘xx’ is the

code used in the branch table. Award columns must be of the form AWARD_xx where ‘xx’ is the code used in the award table. Conflict columns must be of the form WAR_xx, where ‘xx’ is the code used in the conflict table.

Maintenance of CSVUpload

Adding in new ranks, awards, branches, and conflicts is easy. Ranks don’t need to be added manually and any new entries in the .csv under the RANK header will be automatically added to the military_ranks table. One can add in new awards by adding them to the awards table prior to using the .csv upload. The same is true of branches and conflicts; add them respectively to the branch and conflict tables with their respective code matching the last two letters of their column name. For example, if we have a new award with code XY, we’d add a new award with code XY to the database, and then we’d add the header AWARD_XY to the .csv.

Known issues of the CSVUpload

Currently there’s a known issue that can occur if the file upload is large. If the server takes too long to respond, it will throw a 502 error back to the website. The file will still

Additional API Endpoints

The previous team defined another set of API endpoints. These endpoints are solely for the website server. The following Figure 5.J. shows these API endpoints in their respective files and sections.

Server.js holds all the created paths and changes the directory for the server and server APIs.

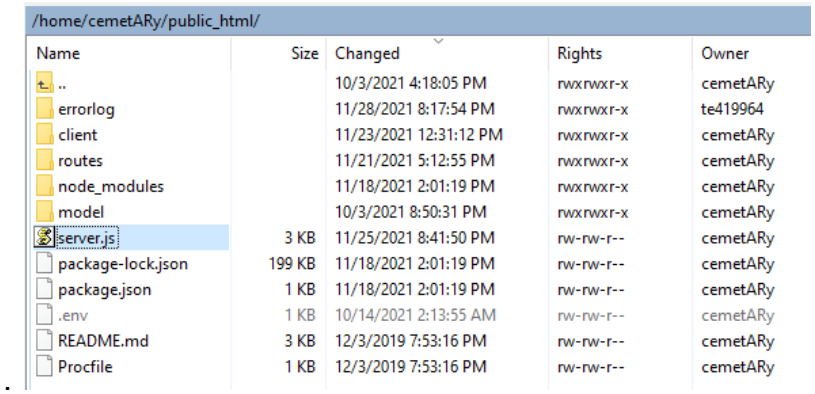


Figure 5.J. - Server.js is located in the public_html folder

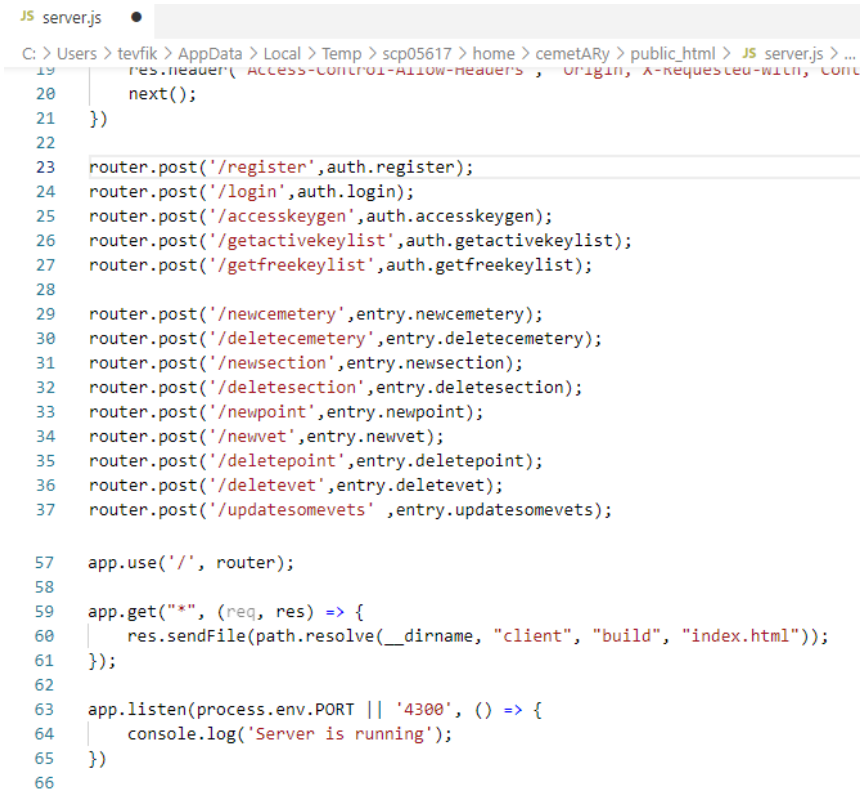


Figure 5.K. - A small portion of Server.js

Additionally server.js allows us to create routers . These routers are used for frontend of the project(website) and mobile portion of the project.
Finally server.js allows us to choose the port numbers and router path.

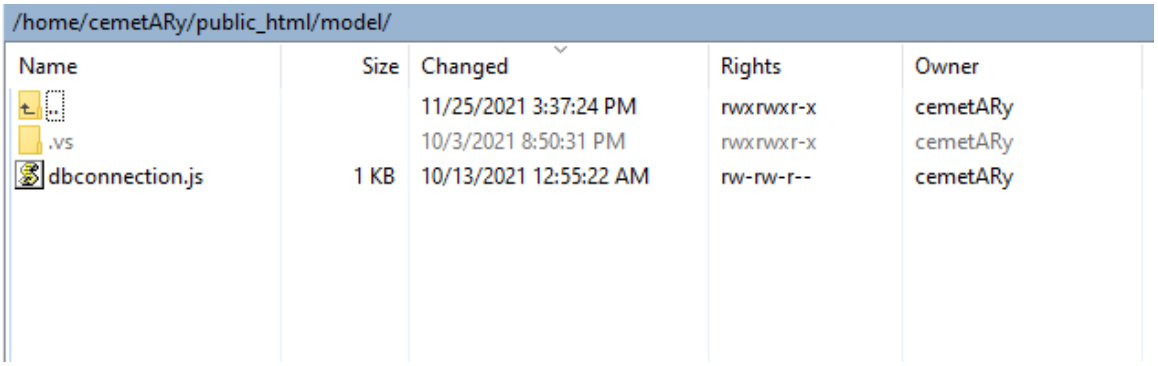


Figure 5.L. - The dbconnection.js is located in the model folder

The MODEL folder holds all the codes for the database connection to the website. When the query is called on the database from the API endpoints, the API creates a connection to the database, executes a query and returns the endpoints to where they should go.

The client folder holds all of the React.js files and requirements about the website. The build folder is created whenever the npm build command is run, and is based off of the files in the src folder.

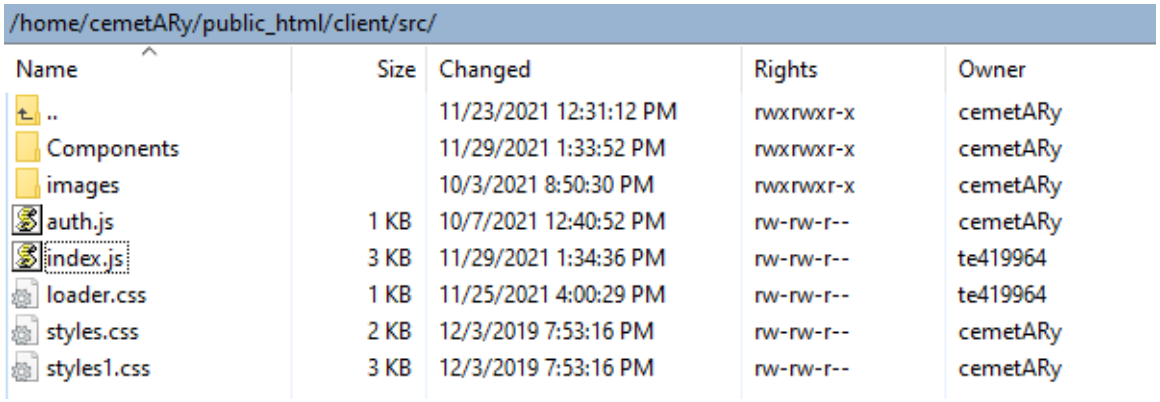


Figure 5.M - The src folder holds all of the files that one needs to edit directly in order to make changes to the website

In the clients folder there is the index.js file that pulls from the Components folder and links the Components to the website. This index.js file allows us to control all of the React components as well as all of the website URLs.

```
import AddPoint from "./Components/AddPoint";
import DeletePoint from "./Components/DeletePoint";
import CSVUpload from "./Components/CSVUpload";
import VetInfoPage from "./Components/VetInfoPage";
import EditVet from "./Components/EditVet";

import auth from "./auth";
import { BrowserRouter, Route, Redirect} from "react-router-dom";

const PrivateRoute = ({ component: Component, ...rest }) => (
  <Route {...rest} render={(props) => (
    auth.authenticated === true
      ? <Component {...props} />
      : <Redirect to='/' />
  )} />
)
//  This is the main function that calls all other components to navigate to the pages of the website.
//  The first path is the sign in page and the other pages are navigated to from there.
function App() {
  return (
    <BrowserRouter>
      <div>
        <Header />

        <div>
          <PrivateRoute path="/dash" component={Dash} />
          <Route exact path="/" component={SignIn} />
          <Route path="/register" component={Register} />
          <PrivateRoute path="/form" component={Form} />
          <PrivateRoute path="/generateKey" component={GenerateKey} />
          <PrivateRoute path="/cemeteries" component={Cemeteries} exact />
          <PrivateRoute path="/addCemetery" component={AddCem} />
          <PrivateRoute path="/addSection" component={AddSection} />
          <PrivateRoute path="/addPoint" component={AddPoint} />
          <PrivateRoute path="/deletePoint" component={DeletePoint} />
          <PrivateRoute path="/csvUpload" component={CSVUpload}/>
          <PrivateRoute path="/veteranInfo" component={VetInfoPage}/>
          <PrivateRoute path="/editVeteran" component={EditVet} />
```

Figure 5.N. - Part of the index.js file; it requires imports and controls the website.

Lastly, we have all of our React component files in the Components folder.

/home/cemetARy/public_html/client/src/Components/

Name	Size	Changed	Rights	Owner
..		11/29/2021 12:28:15 PM	rwxrwxr-x	cemetARy
AddCem.js	6 KB	11/27/2021 5:45:43 PM	rw-rw-r--	cemetARy
AddPoint.js	7 KB	11/19/2021 5:46:31 PM	rw-rw-r--	cemetARy
AddSection.js	6 KB	11/25/2021 10:03:20 PM	rw-rw-r--	cemetARy
Cemeteries.js	10 KB	11/29/2021 1:41:26 PM	rw-rw-r--	te419964
CSVUpload.js	6 KB	11/25/2021 9:36:45 PM	rw-rw-r--	te419964
Dash.js	2 KB	11/23/2021 8:27:17 PM	rw-rw-r--	cemetARy
DeletePoint.js	5 KB	11/27/2021 5:47:37 PM	rw-rw-r--	cemetARy
DropDown.js	3 KB	11/25/2021 9:13:21 PM	rw-rw-r--	te419964
Form.js	25 KB	11/29/2021 1:18:08 PM	rw-rw-r--	cemetARy
GenerateKey.js	4 KB	11/19/2021 5:43:01 PM	rw-rw-r--	cemetARy
Header.js	1 KB	12/3/2019 7:53:16 PM	rw-rw-r--	cemetARy
Loader.js	1 KB	11/25/2021 2:41:54 PM	rw-rw-r--	te419964
Nav.js	2 KB	11/25/2021 3:46:58 PM	rw-rw-r--	cemetARy
Register.js	6 KB	10/18/2021 1:53:04 PM	rw-rw-r--	cemetARy
SignIn.js	5 KB	11/27/2021 8:52:39 PM	rw-rw-r--	cemetARy
VetInfo.js	13 KB	11/29/2021 1:43:00 PM	rw-rw-r--	te419964

Figure 5.O. - Here are all of the React Components in the Components folder

All of these components are related to the React functionality. There are a total of 16 .js files, and we've named them simply to match their function.

5.1.8 - Conclusion about the API Endpoints

This project currently has around twenty endpoints. The API endpoints are written in Javascript using the Node.js and Express.js server frameworks. The endpoints' names are similar to their respective database tables and these endpoints collect data from tables to return a list of data.

These lists of data are used to communicate with the database, web server and the mobile app. Our database has many small tables which are used for the web server and mobile version of the project. The many API endpoints using these small tables are of extraordinary help to the users of the mobile app and maintainers of the website.

Also, these API endpoints make matches with related tables preventing misspelled entries of certain data from causing errors in the database. One final benefit of the API endpoints is that they do not allow the duplication of data for different tables.

5.2.10 - Password Encryption

As previously mentioned, our database has an ADMINISTRATORS table that holds the information pertaining to the administrators of the administrative website. The user first registers with the access key generated by the website and sets up a password. Other information is also noted, for instance, the user's first and last name, as well as his or her email. However, the password for the user must also be stored, but not be available publicly to the other administrators. The password needs to be stored within the password section of the ADMINISTRATORS table, but not as plaintext.

The previous team was able to find a solution to having the database save the password without revealing it to others. Our team will continue to use this feature. If our team needs to, we can further improve the feature, but it likely won't be necessary.

Figure 5.P. - Encrypted password section: all other fields are public to other administrators

The previous team used Node.js libraries for encryption to create a hashed password and save the encrypted string into the database. When signing up or logging into the website, the user can save or input their password to login as plaintext. The website takes that plaintext and encrypts the password using the BCrypt libraries. The encrypted password is then saved into the database. The encrypted password becomes public to the other administrators on the website, but is relatively safe knowledge.

The BCrypt helps not only keep the website safe from potentially malicious activity on the part of administrators [1], but also from other non-affiliated people on the web. It can help prevent brute-force styled attacks.

The previous team did have an issue with asynchronicity, but that problem has more or less been solved already. The library can cause issues with the React API because it's too slow, but thankfully by delaying the database query until after the encryption has occurred, no such problems persist now.

```
In pseudocode, the process for encryption is as
follows:
bcrypt.hash( password ) {
// Add user to database
}
// To login
bcrypt.compare(password,hashword){
if(match){// Login}
}
This ensures that the encryption will occur before the database is queried,
assuring error free authentication.
```

Figure 5.Q. - BCrypt pseudo code

5.1.11. Web Tokens

In order to connect the database to the website, this project needs API endpoints. These API endpoints need to be secure and are used for registered administrators only. The previous team used authenticating web service tokens for these two requirements on the back-end of the project.

Authenticating web service tokens that are commonly used for the implementations of these services are JWT, or JSON web tokens. The JWT token will be generated upon logging into the administrator page. The tokens will then be checked, and then the query will be created. If the JWT tokens do not match, they will be read as invalid. The query will return an error code and the user of the website will not be able to access the database.

5.1.12. Building and Testing

Building the Database

The previous team put together a database using the MySQL workbench for each table mentioned in the previous sections on Heroku. Dr. Giroux instructed our team to change the database's server to the UCF server, which basically required us to import the old database over. There were a few errors in the import because the MySQL version of the previous database was outdated, but we were able to fix them.

Dr.Giroux provided us SQL files to reference as blueprints for the database. First of all, we began by importing that file.The server that she provided us uses PhpMyAdmin. PhpMyAdmin has an import function to add databases. We were able to import most of the tables that were on the SQL files' code.

However, there were some difficulties stemming from the fact that the SQL file originated from the Heroku web server. Heroku has its own methods of creating codes and definers. As a result, we had an error message on the PhpMyAdmin while we were importing the tables.

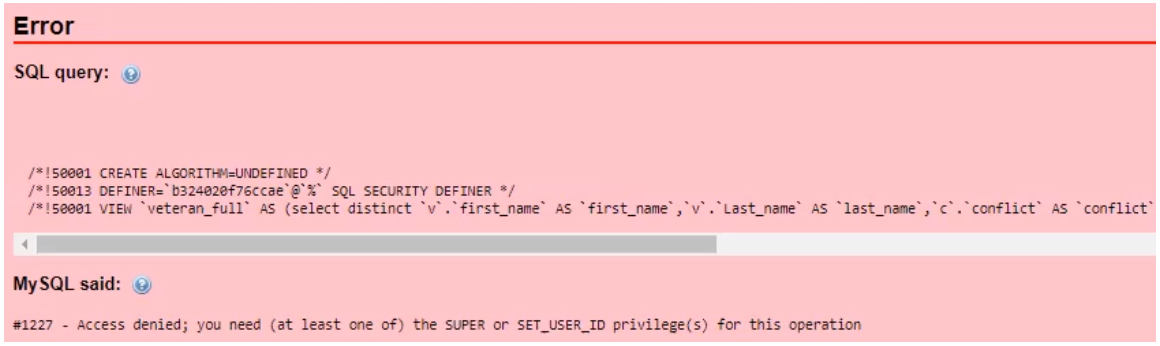


Figure 5.R. - Error message due to veteran_full view table

The error is about the view table called veteran_full. The veteran_full table was created for the mobile portion of the project. This table is just a view table, so it exists solely to help the app obtain data or have easy access to the veteran table.

At the time that we had this error, we did not even know that this particular view table existed. After we talked with our sponsor, she explained to us the contents and the purpose of the table.

After we received the error message and were explained the problem, we focused on solving it.

```
-- Dumping routines for database 'heroku_662c790cea26f36'
--

--
-- Final view structure for view `veteran_full`
--

/*!50001 DROP VIEW IF EXISTS `veteran_full`*/;
/*!50001 SET @saved_cs_client          = @@character_set_client */;
/*!50001 SET @saved_cs_results         = @@character_set_results */;
/*!50001 SET @saved_col_connection     = @@collation_connection */;
/*!50001 SET character_set_client      = utf8 */;
/*!50001 SET character_set_results     = utf8 */;
/*!50001 SET collation_connection      = utf8_general_ci */;
/*!50001 CREATE ALGORITHM=UNDEFINED */
/*!50013 DEFINER=`b324020f76ccae`@`%` SQL SECURITY DEFINER */
/*!50001 VIEW `veteran_full` AS (select distinct `v`.`first_name` AS `first_name`,`v`
`rank` from ((((((((((`veterans` `v` left join `vet_conflict` `vc` on((`v`.`veteran_i
`va`.`veteran_id`))) join `award` `a` on((`va`.`award_id` = `a`.`award_id`))) left jo
`ve` on((`v`.`veteran_id` = `ve`.`veteran_id`))) join `ethnicity` `e` on((`ve`.`ethni
`r`.`rank_id`)))) */;
/*!50001 SET character_set_client      = @saved_cs_client */;
/*!50001 SET character_set_results     = @saved_cs_results */;
/*!50001 SET collation_connection      = @saved_col_connection */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2021-07-09  3:08:53
```

Figure 5.S. - SQL code

Figure 5.M. is the actual code that our sponsor provided us with from the previous team. This code comes from the Heroku SQL file, and it has different definers from the PhpMyAdmin. PhpMyAdmin can not understand Heroku's unique definers in order to solve this error. After we discussed the problem with our sponsor, we changed the code and deleted the definers, allowing PhpMyAdmin to import the code.

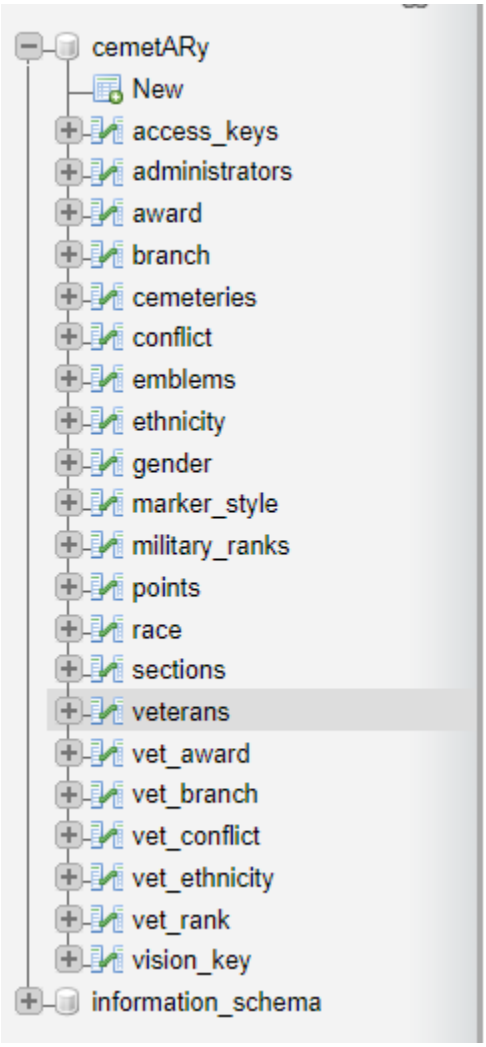


Figure 5.T. - The current version of database tables.

After solving this error, the database became fully functional on the UCF Server.

We were able to complete the database portion by the halfway point of the first semester, which was our goal. We were also able to make some small additions

to the database as per our sponsor's request. Any more changes will be further noted here and will likely occur over the next semester.

Final changes to the Database:

We decided to delete the veteran full view table from the database. We were not using the veteran full table anywhere in the node server or React portion of the website, and it was causing errors. If there is a need for future implementation, it can be easily recreated. We also changed the names of the 'row' column of the veterans table to 'cemetery_row' and 'rank' table to 'military_ranks' because both 'row' and 'rank' are keywords in MySQL.

5.1.13. Database Maintenance

Since we changed the database's server from Heroku to PhpMyAdmin, maintenance is relatively easy. Now that we are finished with the project, there are three things the maintainers should look up.

There will be password encryption on the administrator table. At this point, all we have comes from the previous team project. Since we are not setting up the encryption with our website to our new database, we are unable to test if it is working or not. Our plan is to use the previous team's API for encryption, though we might change that plan in the future.

When new maintainers maintain the database, they should be aware that the encryption exists and ensure that the encryption is working, especially in the event that one moves the database from PhpMyAdmin..

The other thing to maintain is the addition of new tables. In the database, there are many small tables helping the veterans table. There are many foreign keys and primary keys in the tables. These keys protect the tables from creating duplicates of the data.

When new maintainers add new tables, they will need to be aware of these keys. Also if one needs to add to an existing table, one may need to delete the keys first and create the same keys after one is done.

```
-- Table structure for table `branch`
--

DROP TABLE IF EXISTS `branch`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `branch` (
  `branch_id` int(11) NOT NULL AUTO_INCREMENT,
  `branch` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`branch_id`),
  UNIQUE KEY `branch_id_UNIQUE` (`branch_id`)
) ENGINE=InnoDB AUTO_INCREMENT=33 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `branch`
--

LOCK TABLES `branch` WRITE;
/*!40000 ALTER TABLE `branch` DISABLE KEYS */;
INSERT INTO `branch` VALUES (2,'US NAVY'),(12,'US AIR FORCE'),(22,'US MARINES'),(32,'US ARMY');
/*!40000 ALTER TABLE `branch` ENABLE KEYS */;
UNLOCK TABLES;
```

Figure 5.U. - SQL code content

The previous team did a great job on the coding. They included drop tables codes of each table like in the figure above. They added the drop function which most of the database allows. This function can be very helpful to move the database from one server to another. The function itself protects the database tables from importing differences. This is to say that when someone makes a change on a different server and wants to add to another version of the database on a different server, one can just easily import without duplicating the tables. When one creates new tables in the database, make sure to also enable the drop function on any new tables created. Doing so will help keep the project running smoothly.

5.1.14. Building the API Connections

In order to build the API connections between the database and the website, we needed to learn a lot. We planned to go over the API endpoints and its construction in the Fall Semester and did so. As a team we learned the features of these technologies. The website and mobile portion of the project had different needs with regard to the database.

We improved the database and other sections as needed through the project process. The unknown part for us was that we changed the database's location

from the previous team, so some of the technology implemented by the Heroku server on the database was different.

5.1.15. Database Testing

Testing the database will be essential to prevent future potential errors. Naturally, testing the database will have to be done from the other sections of the project.

Once we'd finished creating the project, we needed to check all of the API connections to ensure that they were made correctly and were working completely. To make sure the database was fully working, our team added, deleted and changed data from the website, and then checked to see if the database registered all of those changes. The biggest functionality test was to enter an entire cemetery via an uploaded .csv file.

The other part of database testing came from the mobile portion of our project. We used the mobile app and its navigation and saw that we were led correctly toward the headstone that we'd searched for in the application. We also checked to see if the mobile application could collect and display all of the information that should be available to it via its connection to the database.

We were able to finish the project on time. The database is fully operative and all the API's can collect information from the database.

Other important testing was exporting and importing the database. This feature is for backup purposes. If one enters the wrong CSV file or deletes some of the important tables in the database, one can easily import a previous database to ensure the database is working. We tested out these features in the database and it is working as desired.

5.2. Website Design

5.2.1. Website Design Research

This section is dedicated to the initial understanding of our task in altering the existing Heroku website. The website provides a user-friendly interface for administrators to alter the database of the cemetARy project. Its purpose has not changed in the new project, but there are several requirements that need to be met in the new iteration of the project.

Our first concern was moving it to the correct location. Instead of relying on Heroku, the website is hosted directly on a private development server: CHDR.

There were many concerns that the original website addressed quite well; it was secure, fast, and had the basic functionality it needed to help the administrators communicate with the cemetery databases. Only authorized users were able to access the website at all, and it was very clear who had access to the site at any given time. In terms of security, https was used to protect the administrators and ensure that the information being presented was the actual page and not a hijacked browser of some sort. On top of https, encryption was used to ensure that a user's information wasn't being intercepted by an attacker.

Finally, an added benefit of moving the server off of Heroku and onto the UCF development server was that the site can only be accessed by someone on the UCF campus or via a UCF VPN. There's a lot less potential traffic and a lot less danger against attackers when one has to have VPN access to UCF in order to access the site.

However, even with all these precautions, there were some shortcomings with the website. While it was designed to be aesthetically pleasing, the functionality was a little lacking. Input of data was fairly manual and tedious; in order to add a cemetery, one had to add one row at a time. In order to delete cemeteries, one also had to delete each of a cemetery's sections prior to deleting the cemetery itself. The search functionality was also a bit congested since it was all on one page.

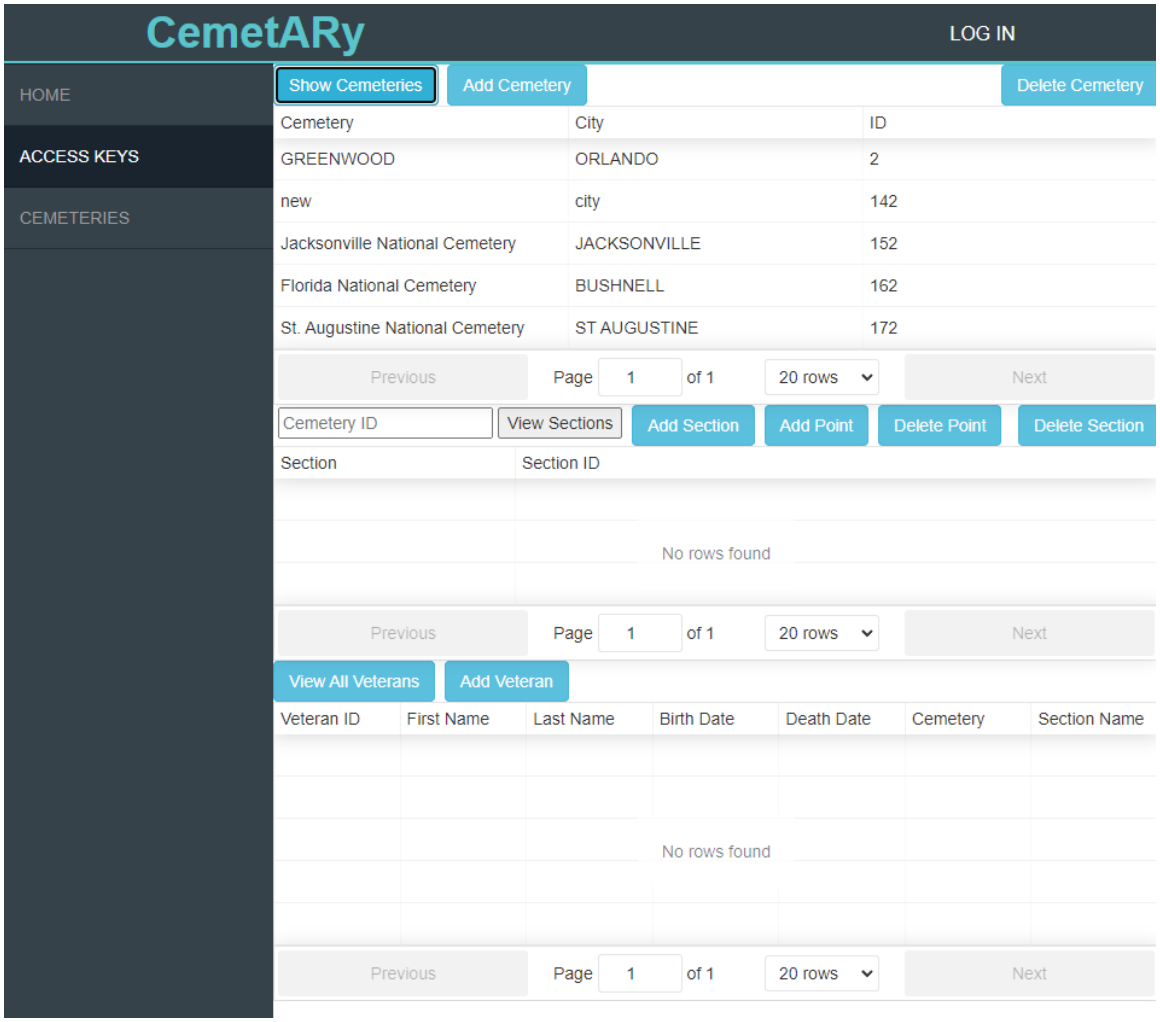


Figure 5.V. - While fairly aesthetically pleasing, the website is dense and hard to parse.

Overhauling the website gave us the opportunity to address these issues. We increased the functionality of the website, allowing us to delete entire cemeteries at a time as well as upload cemeteries via .csv files. The upload function will be on a separate page and show what has changed in the database once the file has been uploaded. Furthermore, we've added edit veteran functionality.

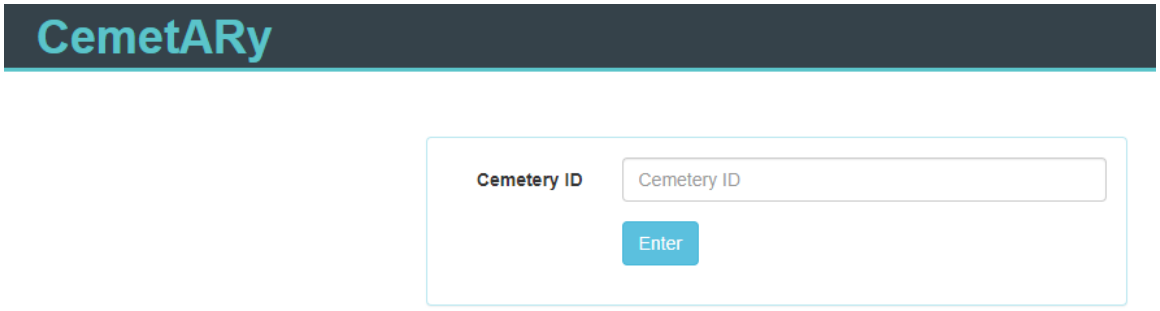


Figure 5.W. - Clicking on 'Delete Cemetery' brings the user to a new page, where the user next needs to input the ID of the cemetery.

One small but important change is the change that we've made to the delete cemetery functionality. Previously, clicking on Delete Cemetery would instruct the user to enter in a Cemetery ID on a new page, which was cumbersome. Furthermore, the delete would fail if the cemetery had any sections still within it. Now, the user can simply click on the delete button next to the cemetery and it will be deleted, even if sections are there.

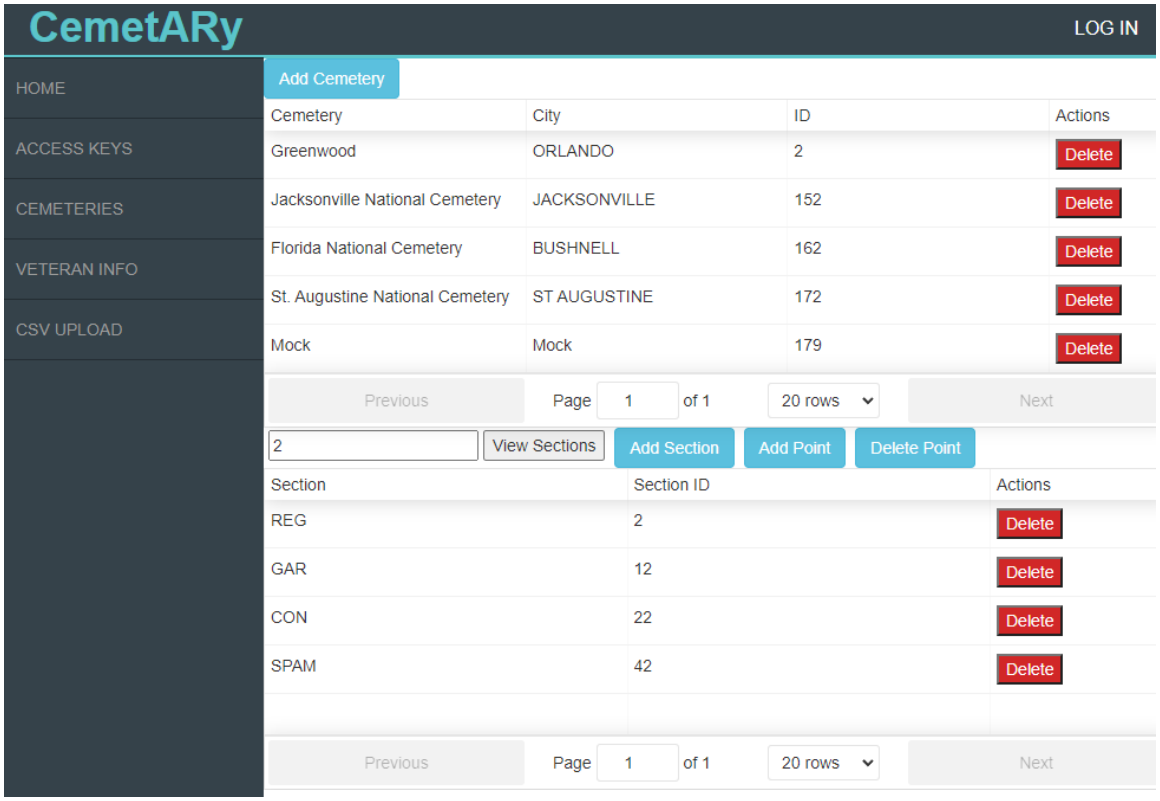


Figure 5.X. - Overhauled Website

Veterans and searching for veterans have been moved to their own section, and a CSV UPLOAD section has also been added.

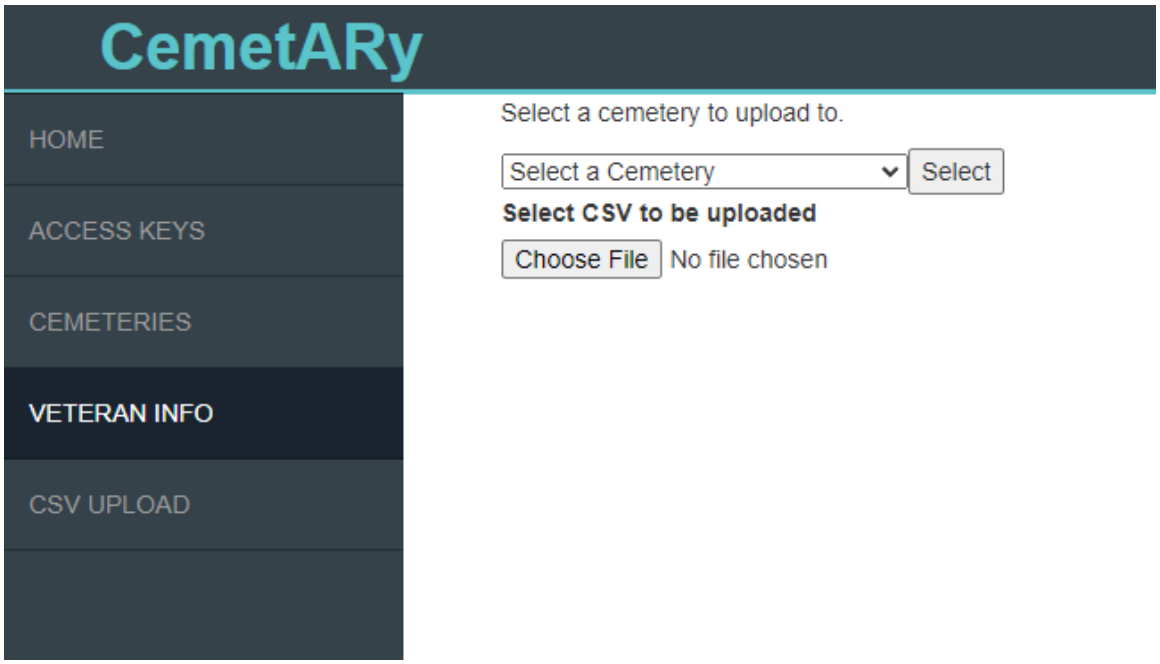


Figure 5.Y. - CSV Upload

The CSV Upload function can parse many veterans at once; details on the CSV Upload functionality can be found in 5.1.7 under the CSVUPLOAD.JS section.

5.2.2. Website Design Specifications

This section will provide an overview of the changes made in the website to meet the requirements of the project. In a lot of regards, the team attempted to recreate many of the things that made the old website possible. The goal was to use CSS style sheets like the previous team did and rely on React 16.9 to create a webpage that is both aesthetically pleasing and simple to use.

5.2.3. Web Application Research

5.2.3.1. React

A large portion of the first semester was spent understanding React, the javascript library used to build the website's user interface. While somewhat

similar to HTML, JSX, the language of React, helps bridge some of the issues of HTML and allows the user easier control over their code by making it function more closely to Javascript [2].

Because not all browsers have compatibility with all versions of Javascript, conversion is necessary. React can rely on built-in Babel.js support to convert its JSX into ES5 Javascript that is compatible with every browser [2].

Beyond simply being highly convenient, React was chosen as our web building library for many other reasons. Its design philosophy as a fast, quickly updating, convenient, and free library felt correct, but there were other considerations as well.

5.2.3.2. React Design Philosophies

React was created in an attempt to design a different way to write JavaScript apps [2]. Specifically, React sought to reduce unnecessary coupling, defined as the *degree to which each program module relies on other program modules* [3]. Naturally, some amount of coupling occurs in almost any large-scale project, but React's developers found that coupling in terms of UI design was a fairly contested subject [4].

The previous best practice was that markup, the appearance of what's on the screen, should be kept separate from logic, the way the page is going to be rendered. This practice stems from the design principle of the separation of concerns, which seeks to divide a computer program into sections that each section deals with one and only one concern. Because markup is a different piece of code entirely from logic, the prevailing thought was that one should keep the two of them separate [4].

However, while in other languages, each step is done separately and combined only at specific points, React's philosophy is that because markup and rendering logic are so closely related, they need to be done at the same time, at the same place in the code. Failure to do so means that one spends time needlessly shuffling between different parts of the code to fix one thing. In other words, they are intrinsically cohesive pieces of code. Increasing cohesion and lowering coupling is the main goal of the separation of concerns, rather than artificially dividing one thing into two [4].

In short, it is the failure to accept that markup and rendering logic are so closely related that causes coupling issues with UI design code. Instead, other web app building platforms tend to rely on view models and templates to provide similar services, but such templates tend to limit their users in what they can do.

React attempts to confront these issues of coupling and eases programming frustration by making them front and center: components are small snippets of code that are self-contained as well as reusable. This reusability is key in simplifying the project overall. Coupling is reduced because components are self-sufficient entities; when something is wrong with the component, only it is affected.

5.2.3.3. Competitors to React

Beyond just React, there were two other main competitors for frameworks that we could choose from. These two frameworks are Angular and Vue. Angular was developed by Google in September 2016 as an open-source web application framework that stemmed from the same people who initially created AngularJS in October of 2010 [5].

Vue, on the other hand, was created by a developer named Evan You and was released in February 2014 [6]. Vue offers perhaps even more options than React in terms of customizability.

Both options seemed tempting and required quite a lot of research in making the best choice for our team.

5.2.3.4. Angular

Angular was another consideration for our team, just as it had been for the previous team working on this project. Angular, a framework for building web applications, uses a language known as TypeScript that is a superset of JavaScript along with HTML in order to create UI. Angular was designed to detect errors more readily and allow refactoring of code with fewer headaches. Furthermore, Angular also ensures better security because it supports structures such as primitives and interfaces [7].

Because TypeScript is a superset, JavaScript will still run perfectly fine on Angular, even if there are some semantic errors that TypeScript disagrees with. In other words, compatibility with older code is very high; it additionally supports features such as decorators or async/await, key functions for the administrative website application that we're trying to build [7].

Angular refers to its components as directives, and keeps very separate the behaviors of components from their UI. The UI is controlled by directives that mark the DOM elements, while the behavior is controlled by JavaScript code [7].

Akin to React, Angular's components help with reusability and scaling into larger projects overall. Regardless of the language we use, we will be able to make use of the component features and all of the cohesion that comes with it, which is a bit of a relief.

Angular's customizability is likewise very high. TypeScript can be debugged in either an editor or even directly in the browser, and one can even refuse to use some of its features if one needs to. These features are particularly enticing to newer users, as some of the options might be too advanced to get into and having debugging on-hand directly in the browser is a very convenient option[7].

The debugging tends to be fairly straightforward as well, with all of the errors that occur being very well-documented. There are a lot of resources to help get one's self out of a bad error [7].

Angular's testing is also very robust and easy to use. Each of Angular's modules can be subjected to automated testing [7]. This ease of testing can be a godsend when testing complicated, interconnected systems like the ones that we have. Having a mobile application, an administrative website, and a database that all need to seamlessly communicate with one another can be a recipe for disaster if we're not careful.

Angular is quite a popular framework, and its predecessor AngularJS has been around since 2010 [5]. As a result, its practitioners have created a slew of resources that we can follow in order to become more acquainted with the system and its inner workings.

There are other niceties as well when using Angular. They have what they call an ahead-of-time compiler, which converts HTML and TypeScript to JavaScript whenever the application builds. The code completes before the browser loads the web app, ensuring a faster render time. Ahead-of-Time compilers are also typically more secure than a just-in-time compiler [7].

Angular also has a robust computer-line interface that automates the development process. App initialization, configuration and development all can be run easily through the command line, allowing the creation of new projects, new features, and new tests with only a few lines of code. As a result of this robust CLI, savvy users can create new applications very, very rapidly. If we become good at Angular, we, too, might be able to speed up production of applications [7].

Beyond the CLI, Angular 6 introduced the Ivy Renderer, which takes a web application's components and templates and turns it into JavaScript that a browser can understand. When the application is rendering, the Ivy Renderer will remove code that isn't used in order to make it more efficient [7].

Out of the box, Angular is already quite powerful. There's no need for third-party libraries to create most applications, and the environment provided by Angular helps both development and testing. Angular as a result of not relying on other parties to create their code is quite secure and has fairly high code quality [7].

Angular also has the benefit of just using plain old JavaScript Objects over something like JSX which attempts to read more like HTML. It can be nice to know that even in a new setting, one can be secure in their fundamentals of JavaScript and be able to understand how certain objects will behave.

The code that Angular tends to create is also highly documented and designed. There is only one 'best practice' way to create a particular component or service or module. As a result, good code created for Angular is highly consistent and will be easy to pass onto other teams in the future.

The CemetARy team has previously had not one, but two teams, and consistency would have been greatly appreciated. It can be difficult to pick up a project during the best of times, but perhaps Angular might have been very welcome to our group had the previous teams used it [8].

As a result of being more consistent, Angular generally provides a fair productivity boost as well [8]. Even within the same team, developers can have difficulty understanding each other's code with other libraries.

That said, because our team is very much divided into different parts of the project, with one person working on the website, one person working on the database, and two people working on the mobile application, perhaps this point matters a little less for us. Still, it'd be nice for us to create consistent code that others can easily pick up without a large explanation, and of the three frameworks examined, only Angular really has defined such guidelines for their language.

One final benefit of Angular is that it is developed and supported directly by Google, which should essentially ensure that the platform has a very long shelf-life; Google tends to create long-term plans for their products and states exactly when said plans expire.

As a slightly tangential benefit, Angular because Angular is supported by Google, it has high relevance outside of our project as well for future projects with a larger audience. Angular is used by PayPal and Samsung and many, many other very large companies, so knowing the framework will be useful in future contexts as well as present.

Despite all of these advantages, however, Angular, unlike both Vue and React, utilizes a real Document Object Model (DOM) directly, and rebuilds the entire DOM each time a change is made to the state. This rebuild takes significantly more time than using a virtual DOM, which is what both Vue and React use [2].

A virtual DOM represents a DOM object but doesn't actually change what's viewed on screen. React handles the problem of dealing with the real DOM by updating all of the virtual DOMs whenever a piece of JSX is rendered. Then it compares the virtual DOM with a snapshot of the same DOM taken before the update was completed. This comparison, known as "diffing", tells React exactly what objects have changed, and then simply changes those objects and only those objects on the real DOM [2].

Angular has been cited as difficult to learn and complex, despite their developers' best efforts to lighten the load for beginners. Furthermore, Angular has the reputation for being tedious to code. One may need to write many, many lines of code just to have a small component appear. Some things that can be programmed instantly in React or Vue may take whole blocks of code in order to render in Angular. Even just setting dependencies can be difficult, and backwards compatibility with AngularJS is also considered to be very difficult [7].

Despite the many benefits it brings to the table, we decided not to use Angular due to its complexity and due to its reliance on the real DOM. We figured that because the website we were building was not too great in scale, it'd be best to have a lighter-weight application overall.

5.2.3.5. Vue

Finally, Vue was also a consideration as a framework. Akin to React, UI and behavior are both combined as a part of a component. Unlike React, however, Vue tends to combine its CSS files into the components themselves, leading to an even more holistically combined approach than even React, which combines only the logic and the markup [6].

Because the whole point of CSS is to create a unified styling template, code tends to be repeated a fair bit more in Vue since each component must have its CSS file. However, having the CSS right in the component itself also means that the component is completely self-contained; one doesn't need to stray to another part of the project in order to check if the CSS is working correctly. In other words, it is the logical conclusion to React's mission statement that UI and markup should be united, rather than separated.

Furthermore, the inner workings of Vue are slightly different. In React, whenever a component's state changes, the entire component subtree needs to rerender. This rerendering can take a lot of time and can be a bit of a hassle to avoid; to stop rerendering children components, one needs to use PureComponent or shouldComponentUpdate [9].

Even these solutions have some issues, though; both PureComponent and shouldComponentUpdate function on the premise that the subtree's output is determined by the props located in the subtree's root component. Should the

subtree actually be determined by other props, then pureComponent and shouldComponentUpdate will actually create incorrect DOM states [9].

By contrast, Vue tracks a component's dependencies whenever a component renders, so it sidesteps the problem of requiring the entire component subtree to rerender. Essentially, it's as if the components all have shouldComponentUpdate, without the problem of creating incorrect DOM states due to the props not originating from the subtree's root component [9].

In other words, the difference between the two means essentially that when working in Vue, one doesn't need to worry about the suite of performance optimizations available in React, because all of them are done automatically[9].

In terms of languages, Vue uses HTML and CSS as well as JavaScript. The use of multiple languages has its benefits and drawbacks as well. React does have some CSS, but actually a lot of it can be managed inside of the JSX portions [9].

This solitary JSX-for-everything approach gives React unity throughout the entire application and has the added benefit of simply being a full programming language [9]. One can have temporary variables and all the other niceties that come with having a full programming language in a space usually reserved for markdown languages. It's also easier to learn JSX for beginners who are only experienced in JavaScript over needing to learn an additional language like HTML.

That said, in Vue's defense, JSX is no free lunch either in the regards that React in general is fairly complicated to learn. By contrast, for developers who are experienced in HTML, Vue will probably feel easier to pick up. Vue also has HTML-based templates that make it easy to migrate things that exist already over to Vue [9].

The argument for which language is easier to learn is definitely a subjective one. On the one hand, learning HTML entirely just to be able to write templates is definitely not free. On the other hand, one must learn some amount of JSX, which is a syntax entirely unique to React, in order to program in React.

Vue's developers argue that their usage of their domain-specific language makes more things possible with less code, and so is potentially easier to pick up than using either JSX or the render functions that React relies upon [6].

Both Vue and React can make good use of CSS, but while React can have either the CSS directly in the component or drawn outside of the component, Vue typically supports having the CSS directly embedded in the component itself[9]. It's easy to adjust for either design choice, and likely we'll be taking the design approach that if UI and markup are to be united, CSS should be included too in those components. In this case, perhaps it doesn't matter so much if we have the versatility of React, as we'll probably be putting the CSS directly into the components we create anyway.

For our project, the bigger difference between React and Vue in terms of CSS is that React typically uses JS solutions to allow such feats to occur, so one is still writing JSX-type code instead of CSS directly. The tradeoff is versatility for speed and size; React can make use of JavaScript to empower styling, but usually has to have an increased bundle size and/or runtime to match. Vue by contrast, has very flexible styling with its text styling functions [9].

In terms of state management, both Vue and React offer comparable opportunities. State management solutions are varied and robust for both frameworks, and can even in many cases be shared. Both React and Vue can use Redux, for instance. Vue, in turn, has also developed their own state management solution called Vuex, which they believe to be the best option for their platform [9].

The two platforms do differ in terms of their support of their state management and routing solutions, however. Vue gives official support for their companion libraries and keeps them updated along with their core libraries. Meanwhile, React simply allows their community to keep up their state management solutions[9]. Because React's user base is so large, however, such concerns probably aren't of the greatest import. If our project scales to be of sufficient size, we will likely be able to rely on the community in order to provide us with suitable solutions.

Vue does have a very nice set of tools for beginners, which our team will sorely miss if we do intend to use React. It offers a project generator that creates

automatic project scaffolding as well as has the capacity to prototype components [9]. It has the capacity to customize its project builds and supports both user-built presets as well as a myriad of default options[9]. React by contrast, has very limited configuration during its project generation and only offers a single page template.

That said, it is intentional that React has limited their project creation because they want their project creation to be standardized. Never will a developer need to import someone's build preset because all presets are the same. This limitation might actually be preferable, since it will mean that future developers of this website will have an easier time inheriting it.

One other nicety for beginners that we will miss is that Vue is typically considered easier to pick up. React requires knowledge of many things, (JSX, ES2015+, build systems, etc.) in order to function properly, but Vue proudly states that in order to get started, one need only write a single line of script [9]. While the learning curve may be steep, our developers are up to the task.

Vue also gives the developer more freedom that can in some cases be dangerous. Mutations of data are easier to implement and throw fewer errors, which can make development more buggy but also give more creative license to the developer.

React, by contrast, wants to rerun some of its lifecycle hooks whenever the state of its components change. For this reason, it disallows direct mutation of state because other lifecycle hooks would be disrupted; React will simply in these cases use setState.

This choice, too, is a limitation but a blessing in disguise. There is usually some better way to handle processes than directly mutating the state.

One complaint about Vue coding is that because it's so easy to pick up and because it is a newer, less structured language, coding can ironically become convoluted very quickly. The best practices established by a very fixed language by Angular are nowhere to be seen in Vue, and as a result, there can be a lot of headaches when trying to understand other people's code. Vue can be easy to start initially, but may have difficulties in the long run when the project needs to scale because of burgeoning spaghetti code.

Especially because we're new to all of these languages, it's probably a good idea to pick a language that has a balance between being easy to pick up as well as being tried and tested for a long enough period of time as to establish best practices. We'll definitely try our best to write the best code that we can, but allowing for our inexperience, we'd still like to be guided by our system to write code that we can confidently pass onto future inheritors of our project.

5.2.3.6. Final Decision

Ultimately, the decision to use React was not made lightly. As with our previous team, we continued to use React, even though Vue was very tempting. We decided to use React because ultimately it had the larger community and the greater support overall. If ever we ran into a problem in development, we could be sure that there was not only someone else who had run into a similar problem, but also a teacher who would be willing to guide us through the issues step by step.

The applications we're building won't be the most demanding, either in scale or in usage, and so the finer control that Vue provides isn't of the greatest importance and might actually be a hindrance for us when we try to create simpler applications. The high use of HTML over JSX is actually a disadvantage for our team, as the web developer was not highly familiar with the language going into the project.

Finally, of the three, Angular appeared to be the least appealing for our group. The fairly arduous undertaking of learning TypeScript, combined with Angular's rather large overhead in terms of libraries and systems, did not seem worthwhile for our group to pursue. It felt like anything we wanted out of Angular, React could provide just as well and with probably faster runtime overall.

5.2.3.7. Benefits of Using React

Beyond the comparisons we've drawn to other frameworks, it's a good time to reflect on what React does well.

React doesn't rely on templates that require the user to program in a certain way the way many other languages or libraries do. It uses components to separate

concerns, focusing on increasing cohesion rather than artificially dividing UI and markup. The result is that we create only what we can see and interact with and go from there.

This design decision to have components as the building block of the web application allows React reusability. It's very easy to take one block of code and use it elsewhere. User input for forms in one part of the website can be used at another point in the website, increasing consistency.

The React developers' mission statement was along the lines of knowing that a framework cannot know best how to separate the developer's concerns for them. There are simply too many varied applications that have different requirements for that to be the case. Instead of trying to accomplish that impossibility, the React developers decided to provide powerful, expressive tools so that the user could do so correctly.

When the data changes within a component, React rerenders the component and any affected components, rather than the entire DOM by searching through the differences between the virtual DOM and a snapshot of the virtual DOM taken before the change. This operation appears at first expensive, but is actually much faster than dealing with the entire DOM at once.

The result is simply that React is lightweight, fast, and has a middle-of-the-road learning curve, all of which make it a very balanced choice for our group.

5.2.4. React Basics

Without going too into detail about all the things that React can do, it's important to note how even the basic structure of React helps further our goals as developers of the CemetARy project.

5.2.4.1. Installation, Folder Designations, Index.js, and Benefits

All React projects must have a fairly large set of installations prior to any actual coding. The installation process creates three folders: a node-modules folder, a public folder, and an src folder, as well as a README, a .gitignore file, and two JSON files labeled package and package-lock.

The node-modules folder contains all of the additional javascript files that allow for very convenient coding of the project itself. The public folder contains the index.html file that will eventually become the home page of the project, and the src contains the source files that house the main code of the project.

The src folder has many potential files within it, but one mandatory file to begin with is the file ‘index.js’. This file imports the other files of the src folder with certain import commands and then links back to the public folder’s ‘index.html’ file with the following line.

```
ReactDOM.render(<App />, document.querySelector('#root'));
```

Without this ‘index.js’, the rest of the project won’t be able to ‘find’ each other. The ‘index.js’ plays a pivotal role in connecting the src folder with the public folder and the index.html.

All in all, the simple configuration of the React project makes it perfect for creating the small administrative website that we need. Navigation is easy and traceable, and anyone who understands React can very readily pick up our project and continue working with it.

Documentation as a result is going to be fairly painless; our React project will look like any other React project, and be accessible to anyone familiar with the React methodology.

5.2.4.2. React Components: Functions and Classes

React uses both functions, which are similar to functions that exist in other programming languages such as C and JavaScript, and classes, which are similar to classes in JavaScript. React functions are called ‘function components’ while React classes are called ‘class components’. A component is simply a reusable piece of code that can be rendered on screen.

The fact that each component has a specific place on the screen means that there’s no real wasted effort; everything is separate but building up to what the user will experience when they use the application. The result is that the development process is both satisfying as well as productive.

5.2.5. Website Specifications

The website has retained its exclusivity and allows administrative access by using an access code system. The previously chosen methodology was to have the user input their name and email into the website, which would then email the administrator a code generated from the backend. The administrator then emails the user the code and the user can then become an administrator. The following UML diagram shows this process:

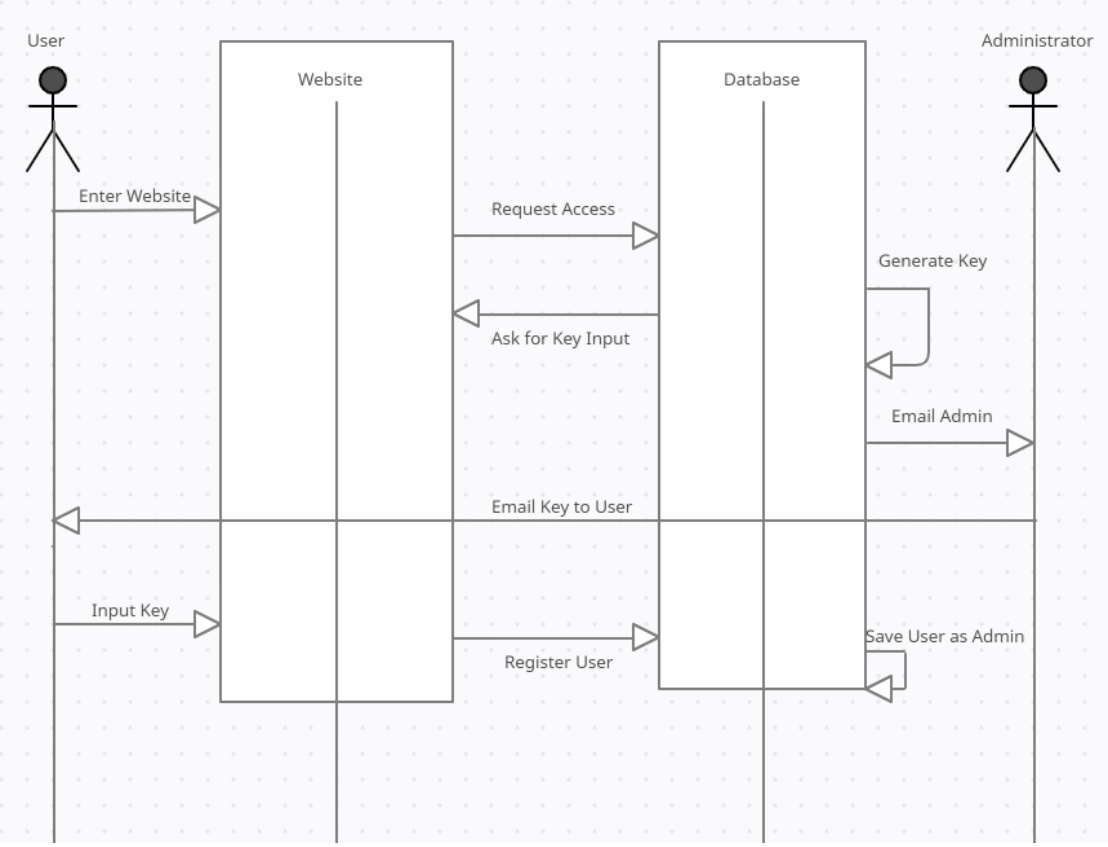


Figure 5.Z. - The chosen key generation methodology

The goals of the website have not changed, either, from the previous team. The website needs to communicate with the database located on phpMyAdmin, and will likely need to use a combination of HTML, JSX, JavaScript, and CSS as well as the React API.

React can set the margins of all of the forms, input boxes, and text where the user puts his or her information and add functionality to the website. When clicking on the sign-up button, the code will call a function that will take the

information that the user has input and then check it against the access code given to that specific user. If it matches, then the user will be granted entry and administrator rights.

Most of the buttons on the interface have similar inputs, though not all of them need confirmation to do what they need to do. Once a user is logged in, they can freely search the databases without further confirmations.

The same functionality needed to be preserved from the initial project. In short, the web pages must provide forms for the user to enter information on the veterans, including but not limited to their names, rank, awards, serial number, date of birth and death, the wars they've fought in, as well as the GPS coordinates of their headstones.

Upon entering the website, the user will be greeted with a sign-in page, which will give the options to login in as well as send a sign-up request. In order to use the sign-up request, the user must provide his/her name and email. The backend will then email an administrator that a request has been made.

The administrator can then decide whether or not to provide the user the key code. Only once the user has entered in the key code will they be granted access to the cemetery databases. If they input the wrong key code, they will be given an error prompt that says that their code is not valid.

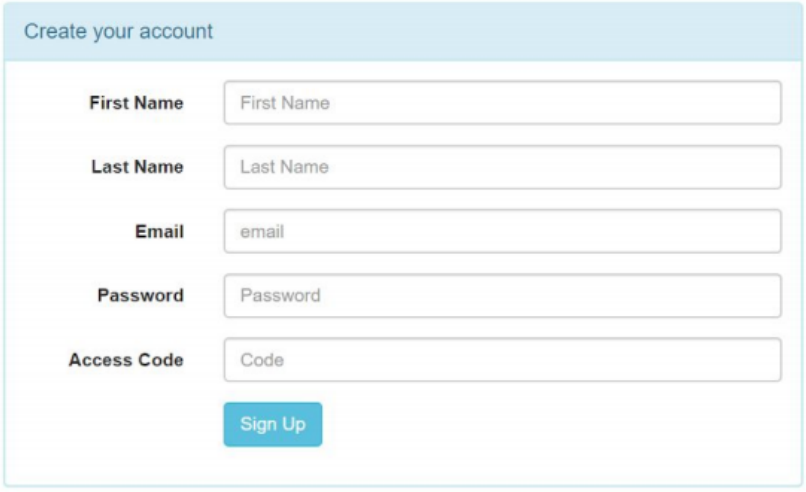


Figure 5.ZA. - Initial form for User Sign-Up

The key codes can be generated via a page available only to existing administrators. The administrators can press a button on the page that will generate a new code and store the code into the database so that a user can be identified with them.

The codes are initially stored in the database and have an attribute that marks them as inactive. When the new user logs into the website for the first time, they will be considered an administrator and the code will become active. All codes are available for look-up in a table tied to the person currently using them, and each person keeps their respective code as long as they are an administrator.

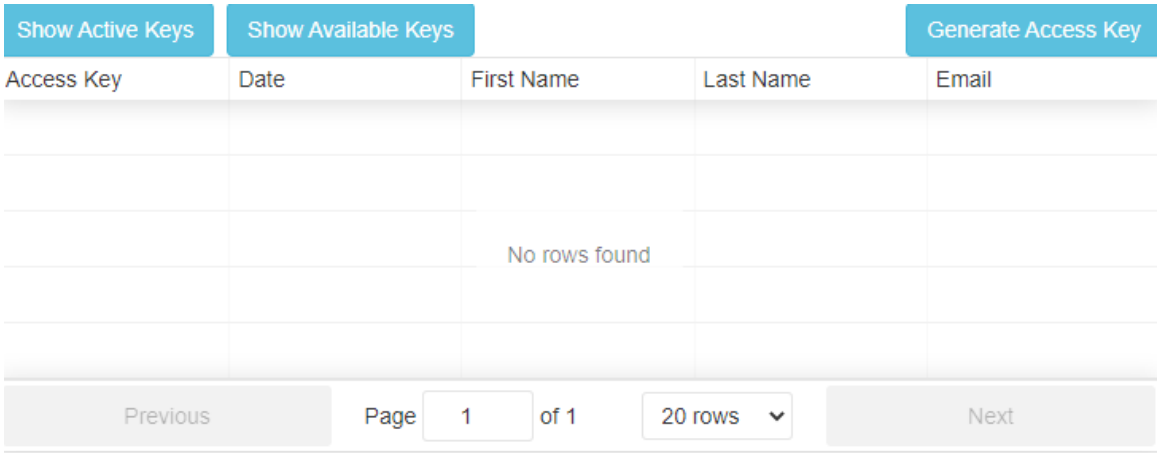


Figure 5.ZB. - Initial form for access key generation

Once the administrator has logged in, they will be greeted by an image of the Greenwood Cemetery and see an interface on the left-hand side that will allow them to select between different options. Currently, they can choose between looking at the home menu, the current access keys, or the cemetery page. Our team will likely diverge from the initial team's setup and split the cemetery page away from a more robust 'search page,' which will allow safe look-up and searching through the existing cemetery pages.

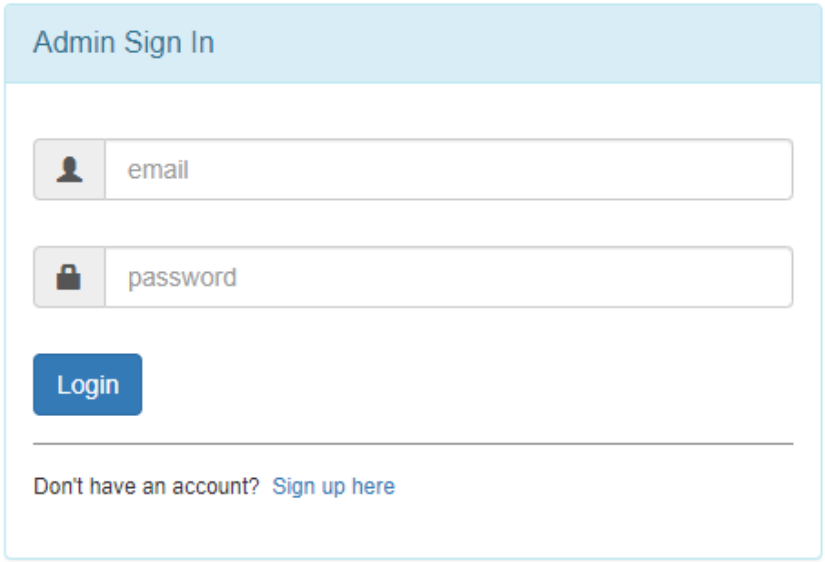


Figure 5.ZC. - Current Admin Sign-in Form

The choice to split the cemetery and the search page in two is threefold. One, having the ability to delete objects right next to a look-up function is very dangerous; if an administrator selects the wrong icon, they can end up deleting an entire cemetery from the database. Two, by splitting the two functions, we can clear up the cemetery page and make it easier to parse through. And three, both the cemetery page and the search page can become more robust and have added functionality without gaining too much clutter.

Besides being able to keep administrators, the website must provide functionality to administrators that will help them manage the database. The previous project allowed users to create individual cemetery objects as well as veteran objects.

Currently, the interface shows a table of cemeteries that have been added and allows for a new cemetery to be added via a button. Pressing the button will open up a menu and allow the user to enter in the cemetery name and city, which will then be added to the list of existing cemeteries.

While we maintained the ability to create individual objects, one of the main goals of the project was to implement a .csv parser that can read-in a .csv file and upload the information to the database. Naturally, it is very dangerous to allow a

parser direct access to a database; if the .csv file is misaligned or improperly constructed in any way, the objects sent to the database may all be incorrect.

Care was given to ensure there's an error logging feature to show if there are any missed entries, and cemeteries can quickly be removed and readded.

After that process, the website will then upload the remaining entries to the database. Node.js has functionality to read through .csv files conveniently, and React has many tools for allowing users to first edit what they've parsed through as well.

Finally, underneath the add cemetery functionality, there will be additional deletion functionality as well. After clicking on a cemetery in the Cemetery List, all of the veterans in that cemetery will populate the bottom table. There will be a checkbox next to each veteran's name that allows for quick and easy deletion of the veteran from the system.

When deleting a cemetery, the administrator will be prompted by a pop-up that will ask if he or she is sure that he or she would like to delete the cemetery. Only after accepting this prompt will the cemetery be deleted. This secondary prompt will ensure that asking to delete the cemetery wasn't a simple misclick. After the command is given, the deletion will be logged and the cemetery deleted.

While the website at first glance seems to be a small portion of the original project, it will actually be the part of the project that administrators will interact with most frequently. As such, it's very important that great care is taken to ensure the interface is user-friendly. The administrators should have no need to access the database directly because they'll be able to control everything they need from just the website alone.

To this end, the search function provides an invaluable function; it allows the user to immediately check if changes to the database have gone through accordingly.

Finally, all of the changes to the database will be logged that can be directly seen and edited by the administrators. This way, if any mistakes are made, at the very least it will not be a difficult task in seeing exactly what went wrong and how to fix the problem.

If there are any additional concerns that the sponsors have with the website, there will be ample time to adjust and deliver a satisfactory product. These current functionalities, (security, the ability to add and delete cemeteries and veterans easily from the database, the ability to log such changes, and the ability to search through the remaining database) are the bare minimum to ensuring a good product.

In terms of user interface, we will have to ask our sponsors to see if they are deemed user friendly, and if they aren't quite up to par, we will adjust the product accordingly.

5.3. Mobile Application Research Process

For researching this project, one of the first things we did was research how the previous senior design team's project functioned. We learned what each of the components were, how they interacted with each other, and how these components functioned in the application.

After doing research on how the old application worked and finding what parts did not work as well as they should or what parts had features missing, we did research on ways to fix these issues. There are many resources online that we used.

While doing this research, we also got in contact with another senior design team. This team is working to help fix issues with the optical character recognition. We had several meetings with this team. These meetings were very helpful, as they both showed us their optical character recognition system and provided us with other resources to help focus our research. They also provided links to both the Github page for their project, as well as pages for other optical character recognition systems.

5.3.1. Unity Application

For this project, the previous senior design team selected to use Unity for the mobile application. There are several reasons for this. One reason for using unity is because it allows us to deploy the application more easily onto both Android and iOS. It also allows for the use of augmented reality, which is an important component of this application. There are also several Unity assets that were used for augmented reality, as well as global positioning data.

Unity is a popular game engine, which can be used for all different sorts of projects. One benefit of using Unity is all of the platforms it can deploy applications to. For our project, we need it to deploy onto iOS devices and Android devices. Unity is able to deploy to both of these platforms.

5.3.2. Front-End Mobile Application

For our mobile application, there are a few things the users should be able to do though the front end of the application. First, there needs to be a feature for searching for information of veterans based on data that the user enters. The mobile application also needs a feature for guiding the users to the tombstone of the veteran that they search for. Users also need to be able to search for information of a veteran based on a photo of the veteran's gravestone.

For our senior design project, one step of research we needed to complete was researching how the previous senior design team implemented these requirements into their project. This way we could figure out how it works, and how we can change what needs to be changed.

Many of these features are already implemented into a Unity mobile application. While deciding what should be changed about the mobile application, one item that was considered was changing the mobile application from Unity to a different software. This is because Unity may drain batteries of phones faster than other mobile application software.

The front end of the project we were given from the previous senior design team is written in Unity, along with the rest of the mobile application. One item our senior design group considered was changing either the entire mobile application or just the front end of the mobile application to use a different SDK. This would help to improve performance and save battery life. Some replacement SDKs we considered using include Flutter and React Native.

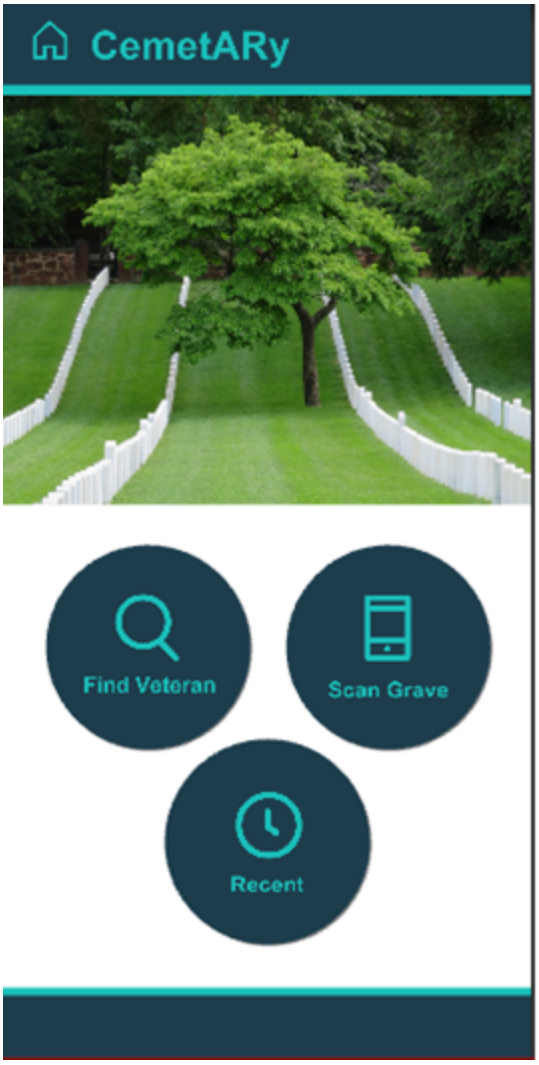


Figure 5.U. - Original mobile application home page

5.3.3. Flutter

One software that we considered using for the web application was Flutter. Flutter is software that can be used for developing native applications for both iOS and Android. Flutter can also be used for web applications as well [10].

The main benefit of using Flutter to develop our mobile application is that it does not use as much energy as Unity. Because of this, the battery usage of our application would be reduced with this new mobile application.

Using Unity, we have GPS data with Mapbox SDK. The Mapbox SDK allows us to track GPS location of users, as well as upload custom data about gravestones.

Mapbox also has a package available for Flutter. Because of this, developing the map system of our mobile application on Flutter may be like the Unity version [11].

One issue with developing our mobile application with Flutter is that our mobile application uses augmented reality. Flutter does have some augmented reality capabilities. Both ARCore and ARKit have available plugins [12, 13]. One of the simplest ways to add augmented reality to your Flutter mobile application is to embed a Unity application into the Flutter application. This way, Flutter would be used for the front end and Unity would be used for the augmented reality. However, the main reason we were looking to changing the application to Flutter would be for the better performance. Because either way we would be using Unity, we decided that continuing to use Unity for the front end may be a more efficient solution.

Computer vision may also be more difficult with Flutter than it would be with Unity. There is an OpenCV plugin available for Flutter [14]. This package, however, may be difficult to implement onto iOS devices.. Unity has OpenCV available as a package. An optical character recognition system would most likely be more simple to implement in Unity [15].

5.3.4. React Native

Another SDK we considered using to recreate the mobile application portion of this project is React Native. React Native would allow us to create our mobile application using JavaScript and deploy to both iOS and Android. The front end of the website portion of this project was written in React, so each part may be similar. Another benefit is that React Native would, like Flutter, give the application a better performance, so battery life will be saved.

Mapbox has an SDK for React Native, just like with Flutter. Because of this, the map and GPS system of a React Native application may be like the map and GPS system of our Unity mobile application. Because of this, it may be easier to implement the map and GPS system for a React Native application [16].

Like Flutter, React Native may be difficult to develop an augmented reality mobile application for. ViroReact is a platform for developing both augmented reality and virtual reality applications with React Native. However, it currently only works with

Android. This is because Apple updated their service to only support applications using WKWebView instead of UIWebView [17]. ViroReact became open-source in 2019 [18]. This was before Apple made that change. Because of this, a simple way to implement augmented reality into a React Native mobile application would be to integrate a Unity application into the React Native application using a package. This may also be difficult to implement, because of the version of Unity we are using. Because the process to develop this mobile application on React Native would be more complicated than remaining on Unity, and we would still be partially dependent on Unity anyways, we decided to keep using Unity for the front end of our mobile application.

5.4. Mobile - Computer Vision

5.4.1. Computer Vision System

For this project, our mobile application must be able to read characters on a gravestone, and then provide data on that veteran. In order to do this, our mobile application must utilize optical character recognition software.

There are two main parts to this optical character recognition: text detection, and text recognition. Text detection is where, from a photo, the optical character recognition system detects where the text is in the photo. Text recognition is where the optical character recognition system determines what characters are written in the detected text. We need to do both of these things in order to correctly analyze what a headstone says and find the veteran that matches.

5.4.2. Current Optical Character Recognition System

One of the main issues with the current project is that the optical character recognition system does not work as well as our sponsor would like. When using this application, users can take a photograph of a tombstone to be analyzed by our application. One important thing for the optical character recognition to perform text detection well is to remove any noise. This means removing anything from the photograph that is not the text. This is part of the detection process.

In the current system, users must manually crop the image by creating a box around the text on the headstone. The boxes also have to be made in a certain order, in order to search for the veteran in the database properly. After the users define the bounds of the text, the optical character recognition system can interpret the text on the headstone.

After taking a photograph, the user must mark on the phone screen by swiping where the text on the tombstone. This is so the noise from around the tombstone can be removed. The photo is then converted into binarized. This means that each pixel of the photo is converted to either black or white. They are converted based on the brightness of the initial pixel. By converting the image this way, the optical character recognition system can figure out where the letters on the headstone are.

The current system uses OpenCV for image processing, and Google Vision to read the words on the tombstone. The words then get searched for in the database. This system does work, but it is sometimes unable to accurately read the tombstone. There are issues with curved text, protruding text, and it is inconvenient to draw the bounds of the text on the headstone.

Initially, the previous senior design team planned to use Tesseract for the optical character recognition. This, however, was changed because of the issues it caused with the rotation of the camera images and the pixel coordinates of the finger swipes. Because of this difficulty, Google Vision was used instead.

Figure 5.V. - Current detection method

One issue that comes up with the system we have is that the optical character recognition system is unable to accurately read curved text. Tombstones with curved text are relatively common, and some graveyards are made up entirely of tombstones with curved text. Many optical character recognition models are designed for reading text documents.

5.4.3. New Detection System

Another senior design team is also working on a system for dealing with this issue. To allow the optical character recognition system to better detect curved text, they created a text detection system that first edits the photo to remove any noise from around the tombstone, then it attempts to straighten out the curved text. Because of this, the system can more accurately read the headstone.

By implementing a new text detection system, our optical character recognition system would no longer require users to draw boxes around the text on the photo they have taken. This would help to make the process much simpler by automating this part of the process.

Figure 5.W. - New detection system by OCR senior design team

Even with this new text detection system, users will still probably get better results if they take good photos of the headstone they are trying to get information on. The text detection system works best when the photo of the gravestone it is detecting text in is a front photo of the gravestone, not at an angle. Having the gravestone at an angle makes the text detection system not work as well. When the user is taking a photo, a guide for how to take the best photo may be helpful.

One issue with this system is that not all tombstones have the same degree of curvature. The text on some tombstones is curved more than text curved on other tombstones. Because of this, the amount of straightening required can vary depending on the specific tombstone. This means that the algorithm developed by the other senior design team may not work on all curved text tombstones. However, this algorithm would still improve the application's ability to read curved text.

Another issue with this system is that it may not work well on tombstones that have protruding text. This is because there is not really a contrast in light between the letters and the area surrounding the letters.

Another feature the other senior design team is working on is an algorithm for deciding which veteran information to receive from the database based on the optical character recognition system's results when analyzing a tombstone. When a tombstone has its text read by the optical character recognition system, they created a system that can calculate a confidence score for each veteran. Then we can select the veteran with the highest confidence score. This may work better than the current system in place. The current system in place returns any possible veteran based on the text it reads. The problem with this is that it depends on knowing what order information is in on the headstone, and some headstones have information in a different order than others. This new system would not be dependent on information being presented in a specific order.

Figure 5.X. - Low contrast headstone.

The other senior design team showed us several models they used for text detection. One model they showed us used PyTesseract for text detection. This

model worked well, but it did not work as well as their model using Google Vision API.

There are some drawbacks to using the Google's Cloud Vision API. One issue is the price. It does cost more than using other computer vision models.

Another issue with Google's Cloud Vision is that it would require the mobile application to be connected to the internet when scanning headstones. One of our goals for this project was to make it less dependent on being connected to the internet. This should not affect how other parts of the application function when not connected to the internet, but it may limit how the mobile application can be used when not connected to the internet.

5.4.4. Barracuda

One way to implement an optical character recognition system into Unity is by using the Barracuda package. This package allows developers to implement Open Neural Network Exchange (ONNX) models to work in Unity. This would allow the mobile application to use computer vision models created using many machine learning frameworks. These frameworks include Keras, Tensorflow, and Pytorch. These frameworks can be converted into ONNX models, so they can be used in our Unity project using Barracuda [19].

Some of the members of our senior design team have had some experience using both Keras and Tensorflow. There are also many online resources for learning these machine learning frameworks, so we did not have difficulty researching these frameworks.

5.4.5. Keras-ocr

Keras does have an option for optical character recognition available called Keras-ocr. Keras-ocr is software that can be used to use and train optical character recognition models [20].

To test keras-ocr, we first used Google Colab to test the model. Google Colab, which is short for Colaboratory, is a service that can host Jupyter notebooks online, and allows users to run python code over their web browser [21]. Using Google Colab allowed us to very easily and quickly use some optical character

recognition models and see how well they performed against the photos we have available.

To test out keras-ocr, we tested how well it was able to read some of the headstone images we have available to us. Some of the curved text images it was able to predict fairly well. In this first example, it read almost all of the text accurately. There are some inaccuracies with the small numbers on the bottom of the image.

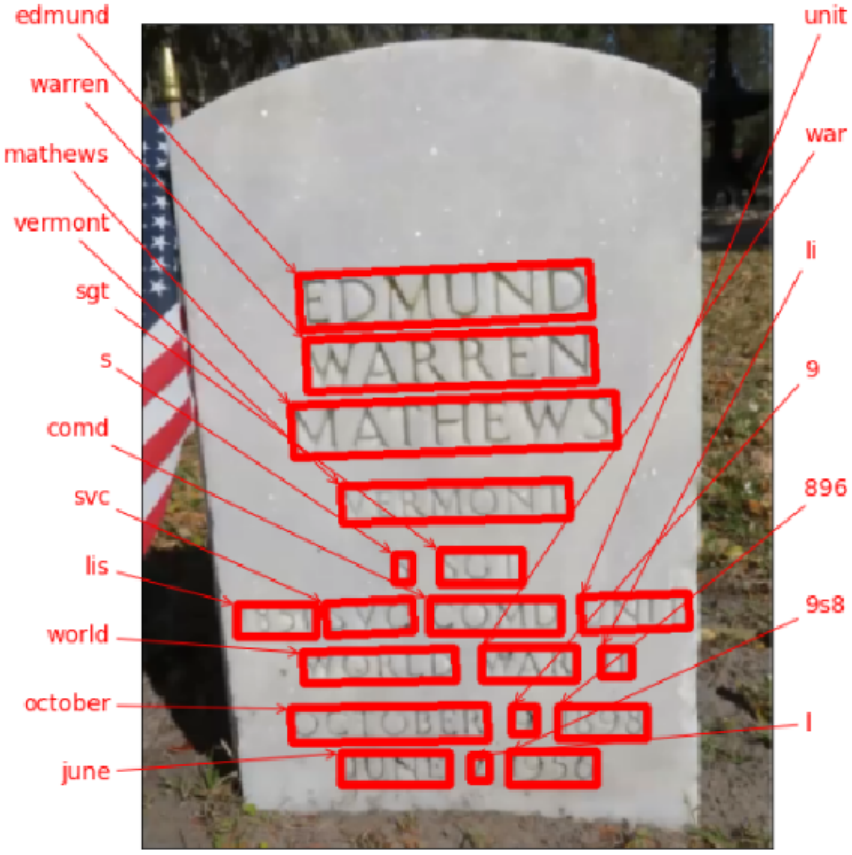


Figure 5.Y - A headstone that is properly read.

In the next example, it read the curved text well. Once again, the only issue with this example's text recognition was when the number text is small at the bottom. This model did however have more difficulty reading some of the more deeply curved text, as well as headstones with protruding text, which do not provide a lot of contrast between the text portions of the image and the non-text portions of the image.

This model seemed to read curved text fairly well. However, because of the issues with other types of text, we continued to look for other possible solutions, which would hopefully provide us better results.

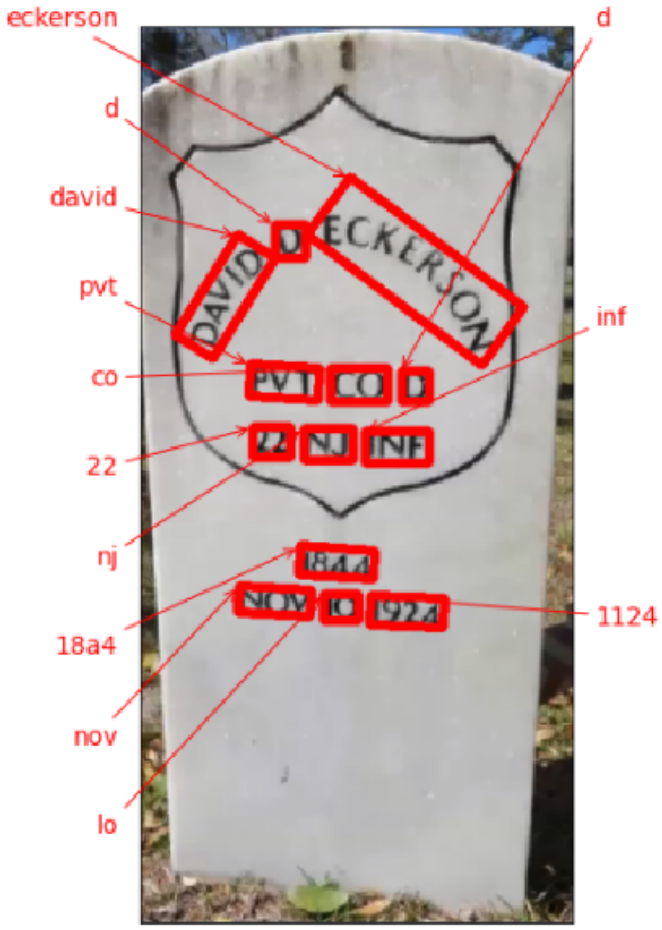


Figure 5.Z. - Another headstone that is properly read, this time with curved text

Figure 5.AA. - A headstone incorrectly read.

5.4.6. Tesseract

Tesseract is an open-source optical character recognition engine currently sponsored by Google [22]. Unlike OpenCV, which is better for preprocessing images than doing optical character recognition, Tesseract is an optical character recognition engine, so it works better for optical character recognition than preprocessing.

The Tesseract API also has a package for Unity, so this is another option for the optical character recognition. There are resources online that showed to us how to use Tesseract, and also how to use it for optical character recognition with OpenCV for image preprocessing.

5.4.7. Google Vision API

Vision API is a service provided by Google. It allows users to easily integrate computer vision and optical character recognition into their application [23]. The current optical character recognition system uses this service to help recognize the text on headstone images.

One benefit of using this service allows the application to be set up quickly. Another benefit of this system is that it works well for text recognition in the current application. Vision's text recognition works better than other models that we and other senior design teams have tested.

The pricing of this system could be a potential issue. Another issue with it is that it would require our application to be connected to the internet to use. Because an application using Vision API would rely on Google, then the system may not work anymore if Google decides to shut down their service.

5.4.8. OpenCV

Our senior design team also did research on OpenCV. OpenCV, which also can be called Open Source Computer Vision Library, is an open-source library. It can be used for image preprocessing, as well as computer vision and machine learning. There are many resources online for learning OpenCV, as it is used in many computer vision projects.

OpenCV is especially useful for image preprocessing. It can be used for optical character recognition, but it is much easier to get a well performing optical character recognition model using a different framework.

OpenCV also has a plugin for Unity. This makes implementing OpenCV into Unity even simpler. For this project, we can use OpenCV for image preprocessing, while another framework is used for the optical character recognition [24].

5.5. Mobile Design

5.5.1. Activity Diagram

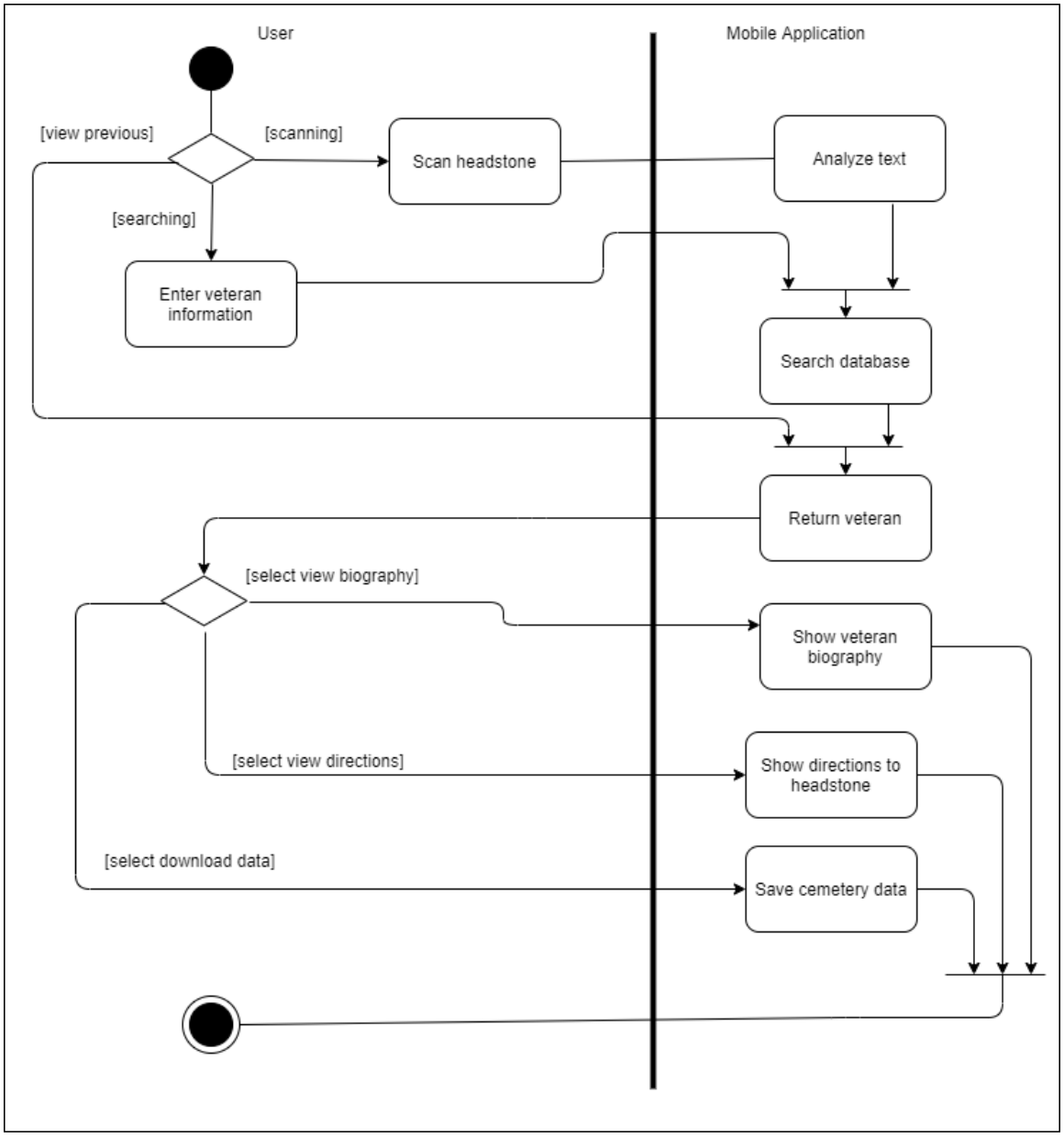


Figure 5.AB. - Activity Diagram for mobile.

In this figure, we can see the activity diagram for the mobile application. There are two groups in this diagram, the users and the mobile application. These are the two main entities that interact with each other in this application.

This activity diagram helps to illustrate all of the actions a user can take with the mobile application and the results of these actions.

5.5.2. Main Menu Design

The Unity project provided to us features a main menu scene. This is the first menu that users will see when opening the application on their mobile device.

There is one button that we will be adding to this project, which is a button for downloading cemetery data locally. This button may be the same as the Recent button, or it could be its own button entirely.

Another feature of the mobile application is a sidebar menu, which can take you to any part of the mobile application. This sidebar can also take the user to an about page, which describes what this mobile application is and how it can be used.

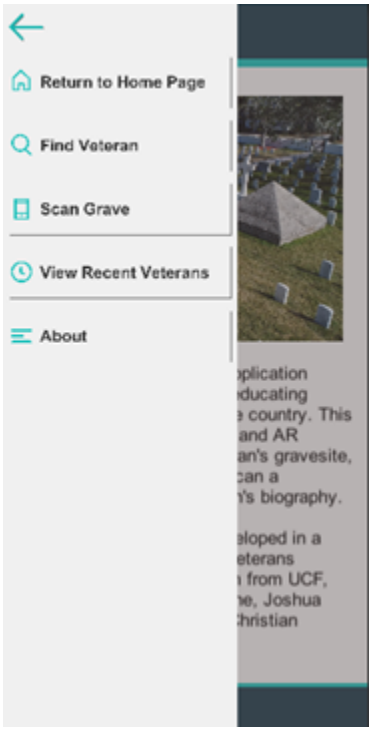


Figure 5.AC. - Sidebar Menu

5.5.3. Search Menu Design

The current application allows users to search for veterans based on certain criteria. The search function of this application which allows users to look up veterans in the database. The criteria a user can search for in the database currently includes the following:

- First name
- Last name
- The state they were enlisted in
- Which conflict they fought in
- What cemetery they are buried in
- Which section of the cemetery they were buried in

The user does not have to enter all of this information for the veteran they are searching for. After the user enters the information of the veteran that they are looking for, the mobile application sends a query to the database using the information the user entered. The database will then send back data of all of the veterans that match the data that the user entered.

There are a few things that we need to do to make this section better. One of these things is to improve the user interface and to help to users more easily search for the veterans they are looking for. Another way to help make this section better is to add more search terms, so they can be more precise with their searches.

Some things we could do to make the user interface more clear is to make the search boxes larger. Possibly having one search term per each row would make the search bar easier to use. Another change would be to make it more clear that entering all of the search terms is not required for searching the database. Users only have to enter the information of a veteran that they know.

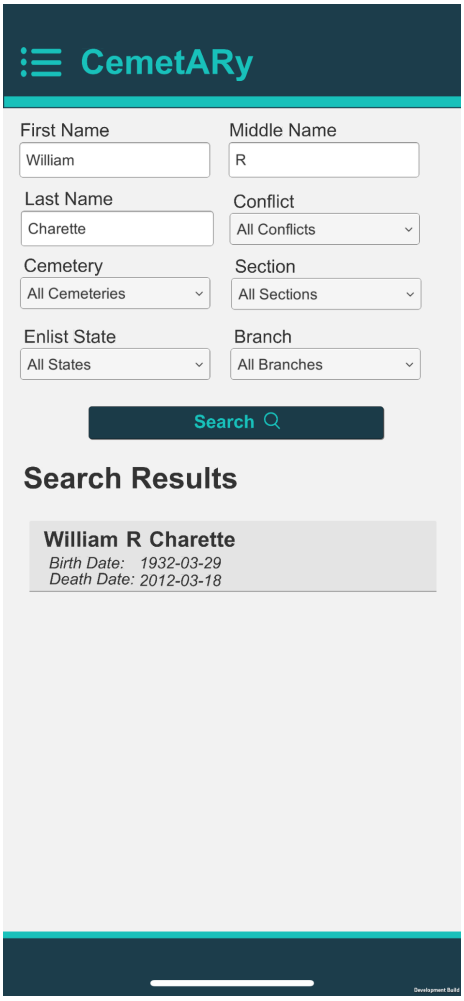


Figure 5.AD. - Current Search screen

However, there are other criteria that users should be able to search for that are not available in the current version of this mobile application. Some new searching criteria include the veteran's middle initial, and the branch of the military they were a part of. With the addition of these new searchable items, the search portion of this mobile application will be more robust, and users will be able to more easily find the veteran they are looking for.

5.5.4. - Veteran Biography Screen

After a user selects a veteran by either using the search function or by scanning a tombstone with the optical character recognition, they are taken to the veteran biography screen. From this screen, they can see the following:

- First Name
- Middle Name
- Surname
- Suffix
- State
- Unit
- Birth Date
- Death Date
- Cemetery
- Site
- Section
- Row

When compared against the search page, the biography page contains much more data. Because we have all of this data available in the database, we could add more search features to the search page. For example, middle name is not a searchable feature for the veterans. We can use the data in the biography screen and add them as terms to the search screen, which would improve the accuracy and precision of the search feature.

For this biography section, we can make some changes to the page to make it more readable to the user. For example, the entries for the data could be spread out more, to make each topic more readable. By making changes to the user interface, we can make the experience for the users more simple and helpful.

The biography section also does not display the row and section correctly. The row is never displayed, even if the veteran has a row. The section name shows an incorrect value. We need to fix these two sections to display the correct information.

Figure 5.AE - Current biography screen

An addition that needs to be added is a link to the Veterans Legacy Memorial. The Veterans Legacy Memorial is a website that provides biographies for veterans in cemeteries managed by the National Cemetery Administration [25]. When a user goes through the steps to find information on a veteran, either by entering the information in the search function or scanning a tombstone with the optical character recognition system, a link to the Veterans Legacy Memorial should be available.

5.5.5. Mobile Application Implementation

For the front end of the mobile application of this project, we continued to use Unity. Each menu the users must be able to use will be different scenes in Unity. This will made development simpler by implementing the new required features into the old Unity project, rather than create a new system that has both the new features and the old features.

The menus of this new system have similar functionality to the old mobile application. However, the menus of the mobile application were redesigned. Some changes were made to be clear on what each menu item does.

One of the reasons we decided to keep using Unity for the front end is because it made development much simpler. No matter which software we used for the front end, the GPS would be easiest to implement with Unity because of its augmented reality support.

Another reason we decided to continue using Unity for this project is our senior design team's familiarity with the game engine. Because members of our team have used Unity in the past, learning the software and implementing the project can be done more quickly than if we were using different tools for developing the front end that we have not used before.

A third reason for using Unity for the front end is that most of this project was already created in Unity by the previous senior design team. Rather than restart development on a different platform, we decided that development would go faster if we continued to use the project developed by the previous senior design team.

All of the front end was developed using Unity. This means that all of the UI elements were created using Unity's canvas. The layout of the page and all of the buttons were created in the Unity engine. Unity has many prebuilt user interface features that a developer can easily put into their project. Using this interface designed in Unity, we can also easily give these buttons functionality with scripts written in C#.

To help us have a better idea of what the new design of this mobile application should look like, we developed a redesign using Figma. Figma is an online tool for designing user interfaces for web applications and mobile applications. This tool has allowed us to more easily plan for what the updated user interface should look like.

Because of how this mobile application was written in Unity, we were able to change the look of the UI without changing most of the functionality. For example, the circular, unnecessarily large buttons in odd locations on the menu were changed to rounded rectangles, which helped the buttons to be more readable, as well as add space for more buttons that are used in our new features.

One other potential change we implemented with these home page buttons is their layout. Since we changed these buttons from circles into another shape, the buttons could be placed in a uniform order. We now have the layout of buttons in a list, top to bottom in a column. With the inclusion of some more buttons, it is an easy way for a user to see the list of options they have with a more standard layout.

A third potential change to the layout of the page is due with the color of the buttons. Right now, they are a green color, which is the same color as the borders of the mobile application. This color of the buttons do contrast with the white background they are in, but changing the color slightly from the outline color may make it more clear that they are buttons that can be pressed. Adding a drop shadow to the buttons may also help with this issue.

5.5.6. Proposed Visual Improvements

During planning, we had proposed the idea to add some visual improvements to the app. Unfortunately, we were not able to implement all of these ideas due to our resources being used elsewhere. Most new visual additions to the app were configuring the app's layout to be more accessible.

We wanted the general look of the app to look more professional. To plan this out, we prototyped a look for the CemetARy app using Figma, a prototyping software. It allows for easy creation of visual elements and adding functionality to those visual elements. Figma projects are also easy to collaborate on, as all changes are immediately saved, and each designer's actions are visible. The prototype created is functional and acts as an example of the desired look of our final product: complete with dropdown menus, scrollable pages, and interactable

buttons. It is navigable in a way similar to how the actual app functions. Its blue and white theming is reminiscent of many U.S. government websites, and the Veteran's Legacy Memorial site [25].

The home screen was made to look sleeker, and includes the new "cemetery pack" option. By selecting it, you can go to a search for the desired cemetery, and download the relevant contents for offline GPS navigation. You can also view cemetery packs you have already downloaded here. The circular buttons on the menu are changed to rounded rectangles, which makes the buttons more readable, uniform, and professional. The layout of the prototype was carried out in the final version of the app.

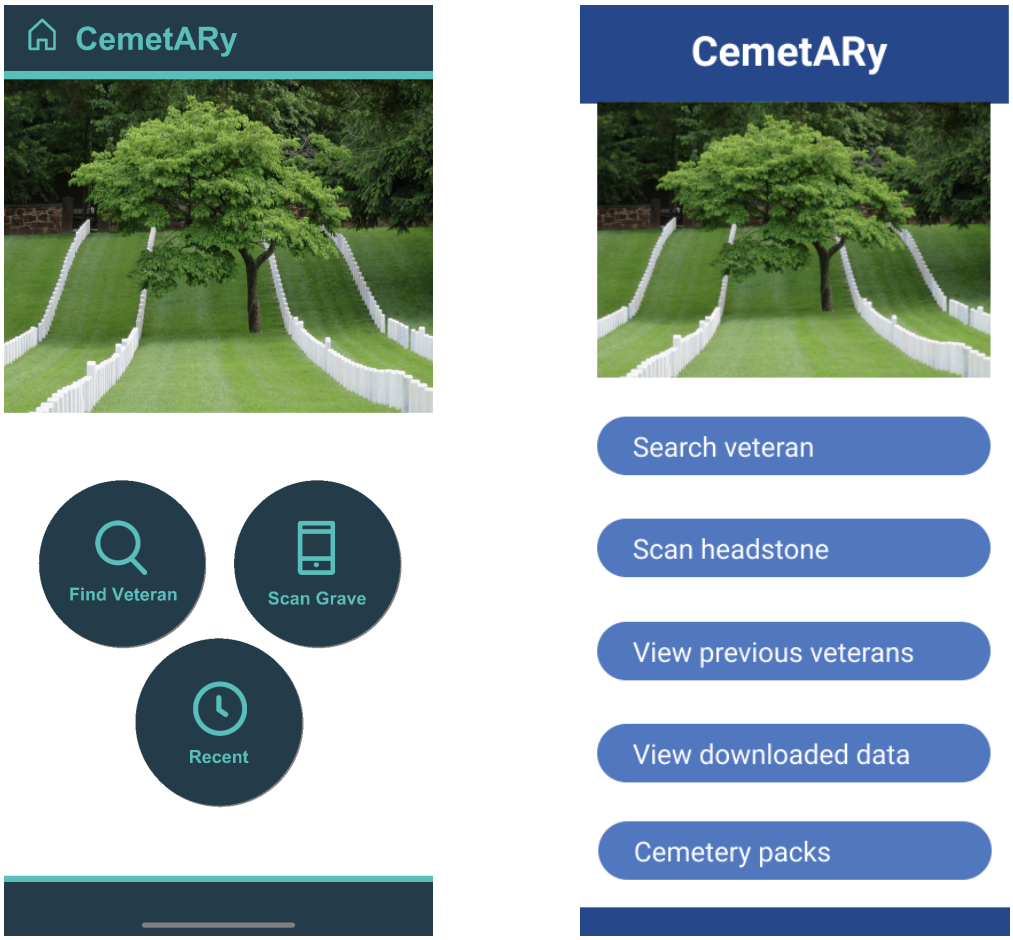


Figure 5.AF and 5.AG - Before and After Redesign

The cemetery pack page is a new addition to the app. It applies the new theme of cemetery. From it, you can select from a dropdown list a one of the supported cemeteries, where it will then show a picture of the cemetery, its address, and a button at the bottom which allows you to download an offline pack of the selected cemetery.

With our design for this page, we want the users to have an easy time selecting the cemetery they would like information downloaded for. Because of this, we want to present all of the information of a cemetery in our selection screen. This includes address, city, state, and ZIP code. By presenting this information, we hope to make it clear which cemetery they are selecting.

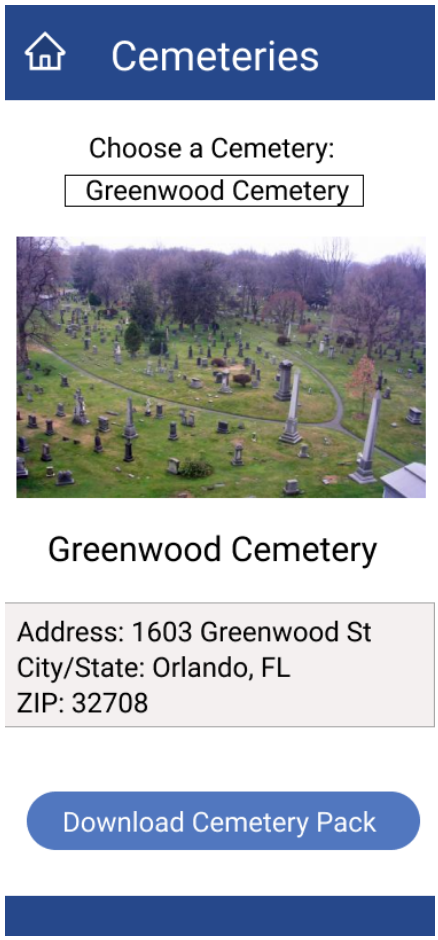


Figure 5.AH. - Figma representation of the Cemetery Pack

When redesigning the About page for our mobile application, the main thing we wanted to do was keep the design of this page consistent with the rest of our application. The old About page was simple and only really had text with an image, so in the updated version we kept these things. The new about page follows the new theming, provides a clearer explanation, and includes additional details about the CemetARy application and development teams.

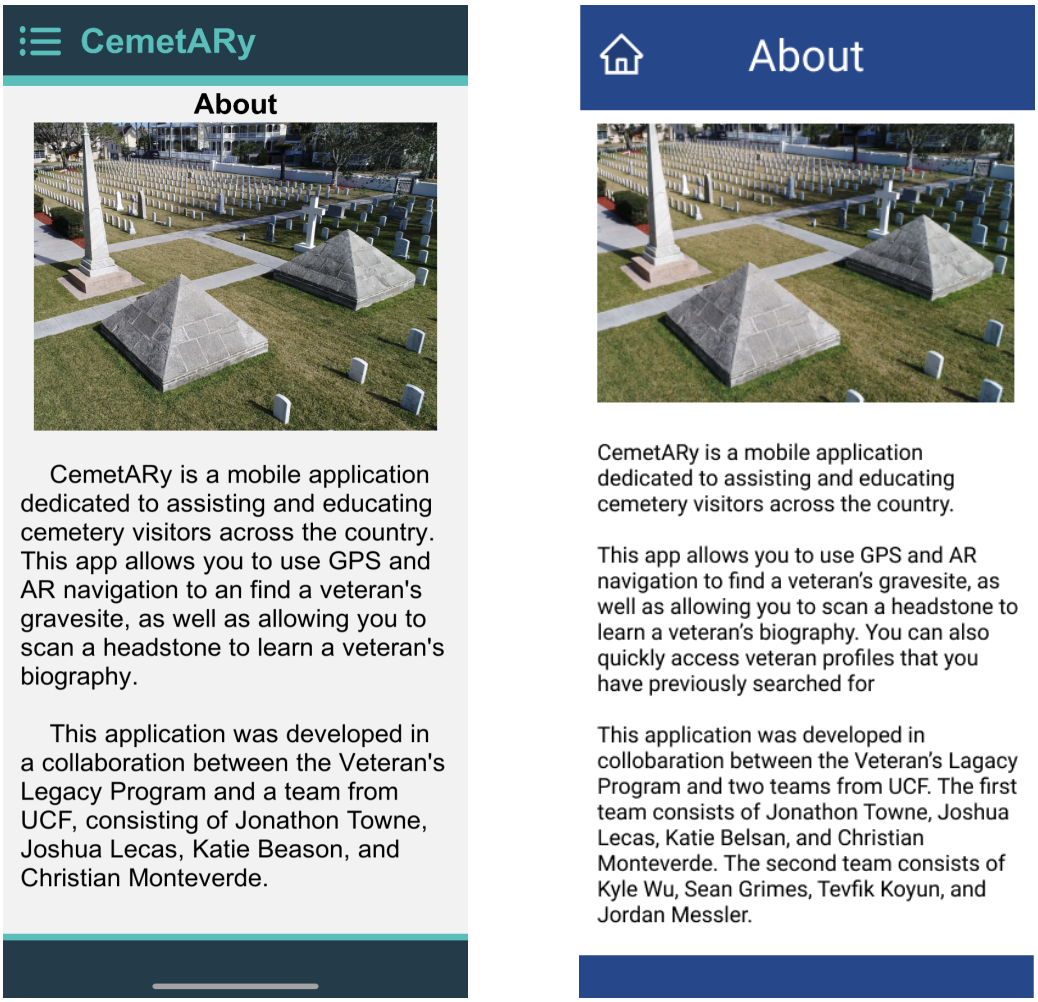


Figure 5.AI and 5.AJ - Proposed Redesign of About Page

The original scanning page has issues with text overlapping the image in an unappealing and unreadable manner. Items and text are offset, and some larger than they should be. The prototype attempts to fix this by completely separating the text from the image, not allowing it to overlap. Unnecessary elements were removed, and the size of certain elements, like the scan complete button, were adjusted to make the page more compact.

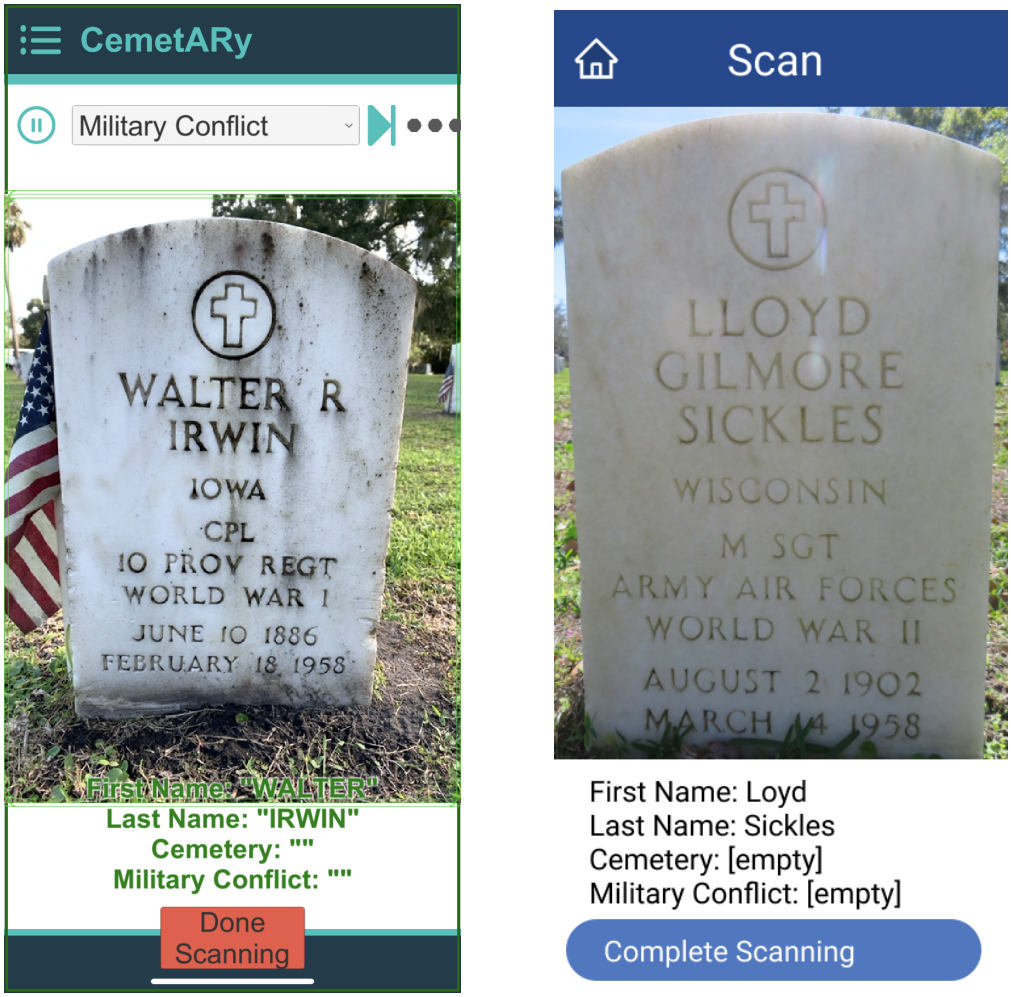


Figure 5.AK and 5.AL - Scanning Visualization Before and After

Due to the nature of the tools used and lack of need for improvement, the GPS screen has stayed relatively the same in both the 2D and AR variations. However, a change of color scheme could do wonders for accessibility. There is enough real estate on the screen to allow larger buttons for easy visibility, as seen in Figure 5AN.

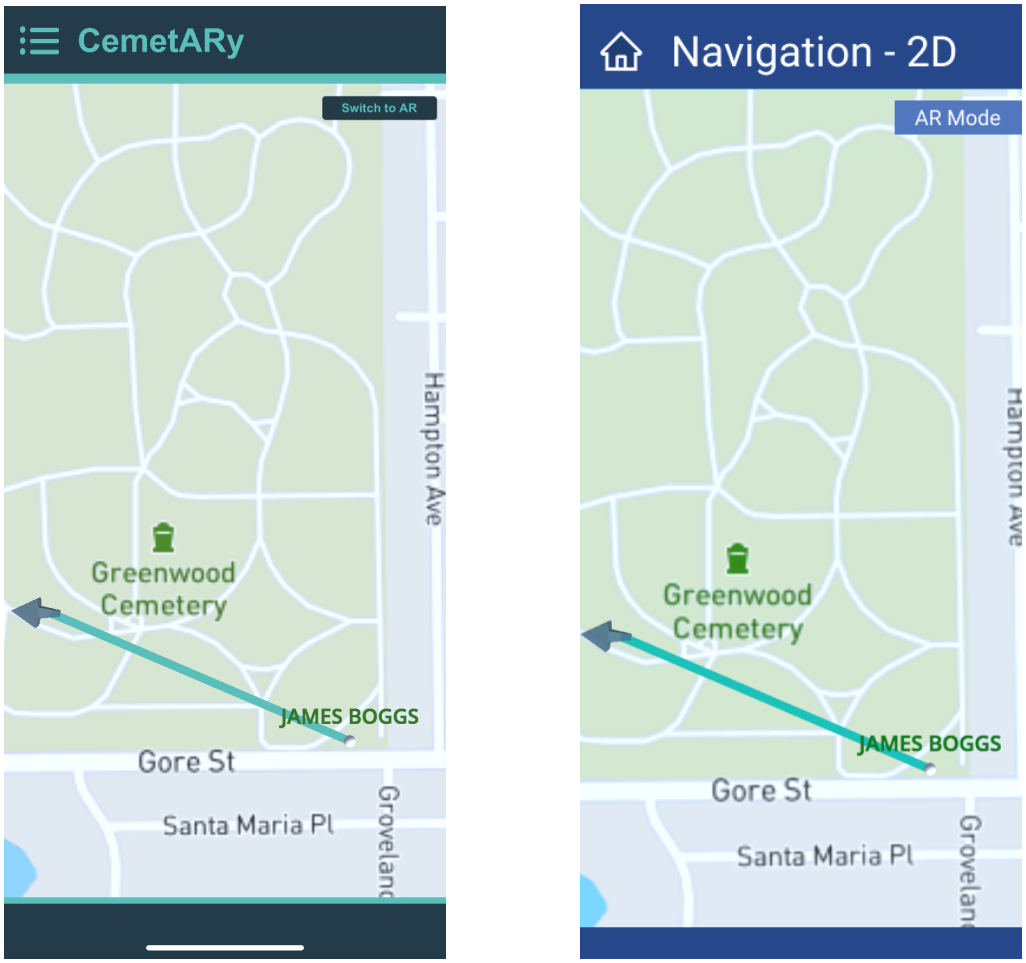


Figure 5.AM and 5.AN. - Top-Down Navigation Before and After

In the AR navigation page's current state, the green, blue-green, and teal colors do not contrast enough with the greenery and dark areas present in cemeteries during the AR navigation sections. Unfortunately, there is not a good gauge of the right coloring scheme to use for the 3D elements, as those are built in Unity. Depending on the background, the contrast can change drastically, such as the dark treetops versus the bright sky as seen in Figure A.P. Text with a bold dark outline is likely necessary for a text readable in all backgrounds.

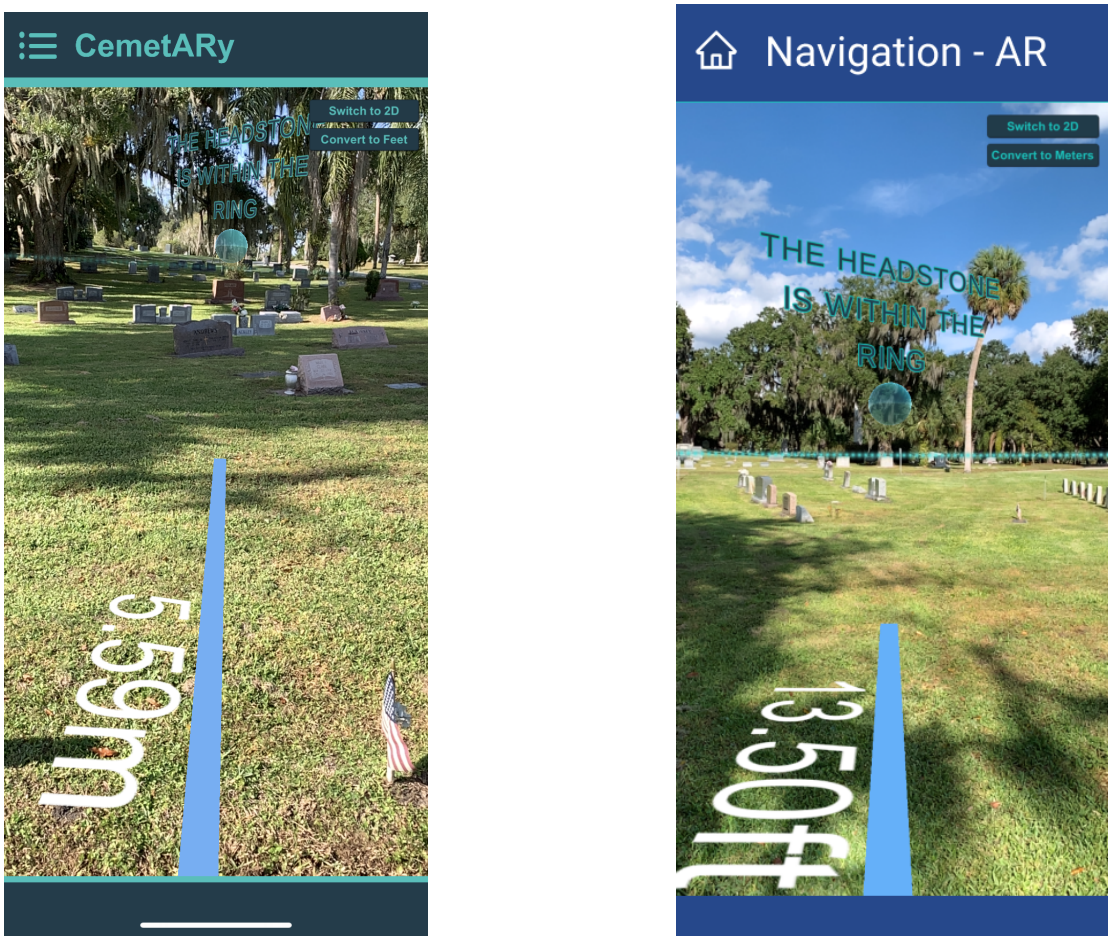


Figure 5.AO. and Figure 5.AP. - AR Navigation Before and After

The veteran profile page was updated to look cleaner, incorporate the new look of the app, and include the new option that links to the veteran's page on the Veteran Legacy Memorial website. Certain elements of the page were combined to make it read easier. The page is also scrollable to avoid clutter.

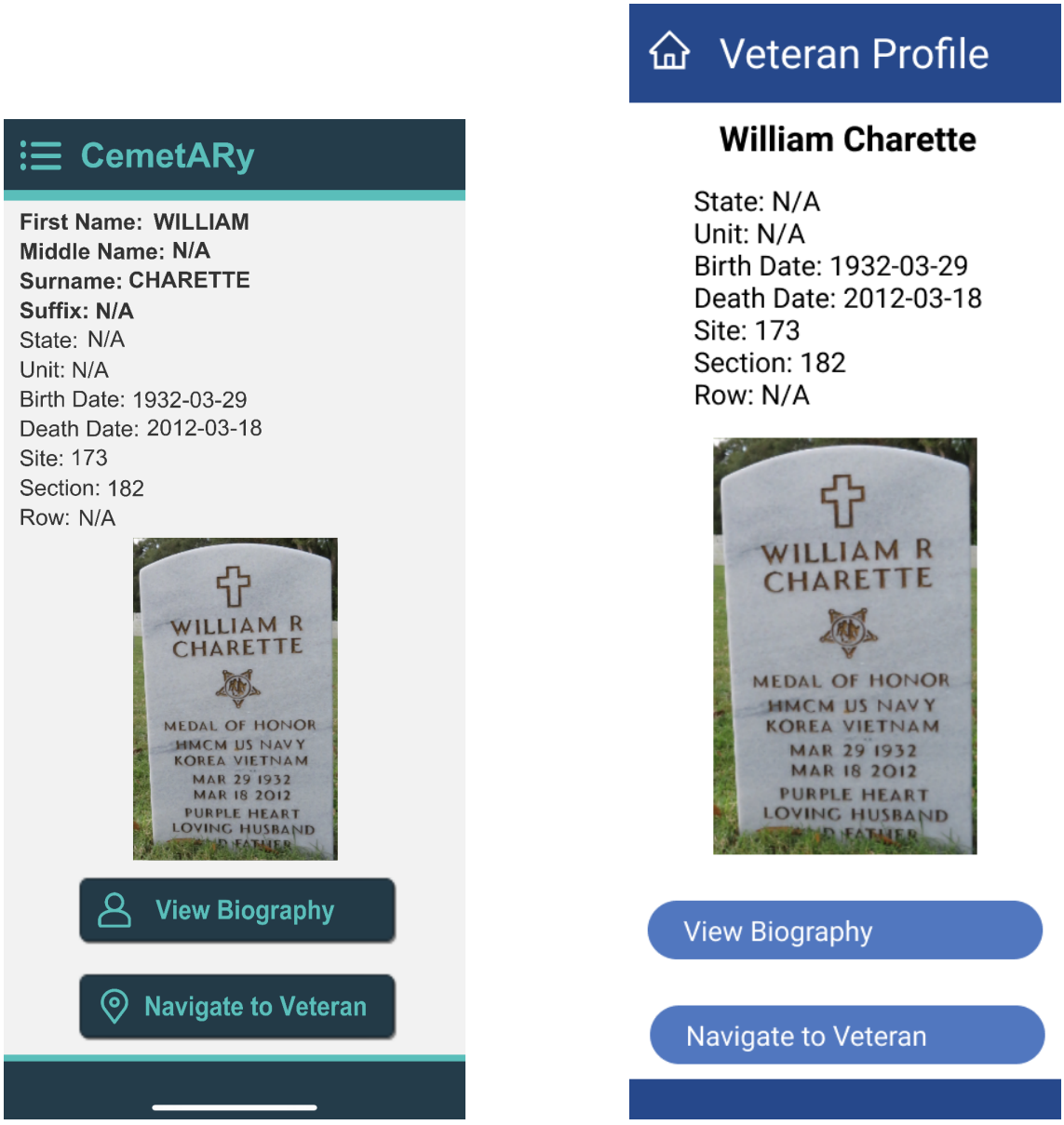


Figure 5.AQ. and Figure 5.AR - Veteran Profile Before and After

The “recently viewed” page is largely the same as it was before. Elements were enlarged to be easier to read, and the page is scrollable. The gray-on-gray background of the original is slightly difficult to read, so it was changed to be gray-on-white.

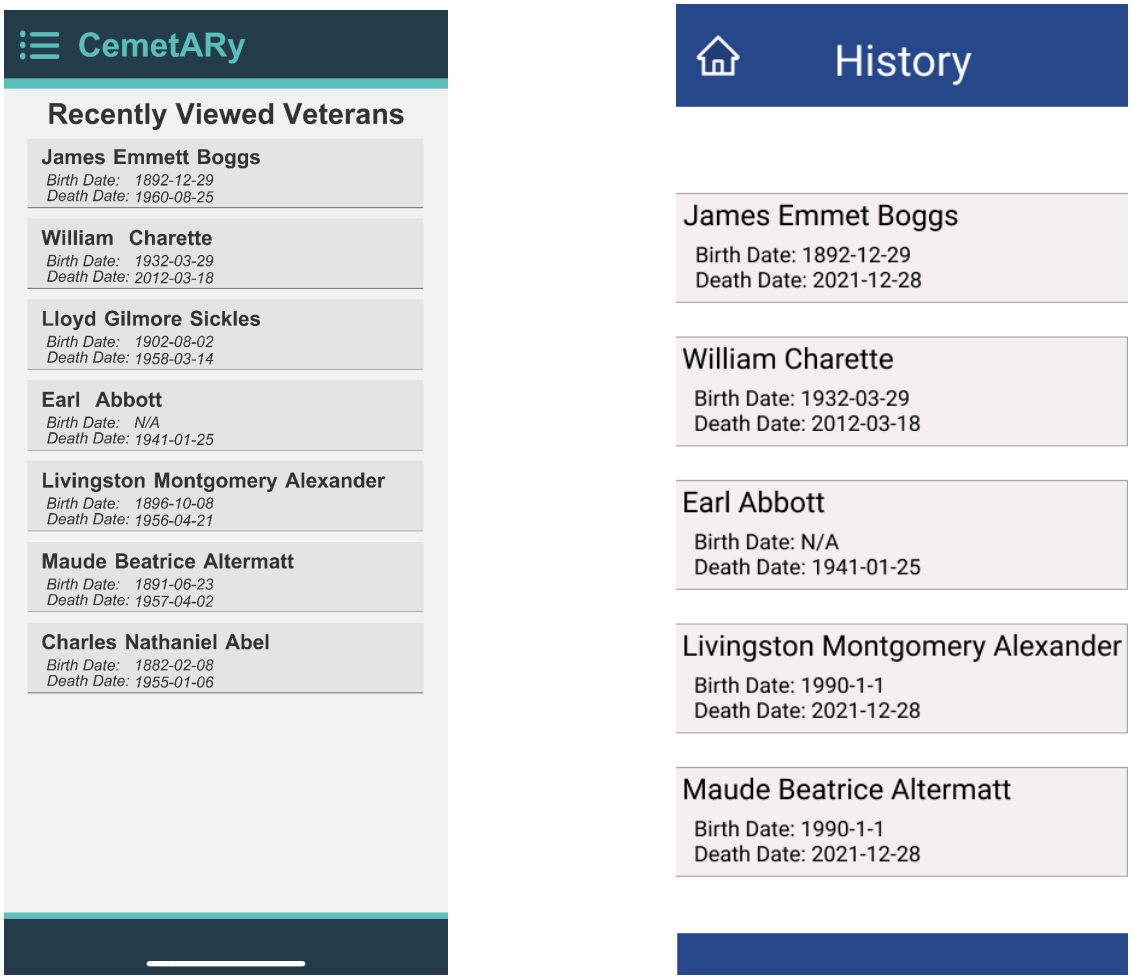


Figure 5.AS. and Figure 5.AT - Recently Viewed Page Before and After

The search page is mostly the same as its last iteration. Elements are larger and the page is scrollable. The state, cemetery, section, and conflict sections retain their dropdown menu capabilities. The gray elements on white background are applied here as well on results to increase visibility.

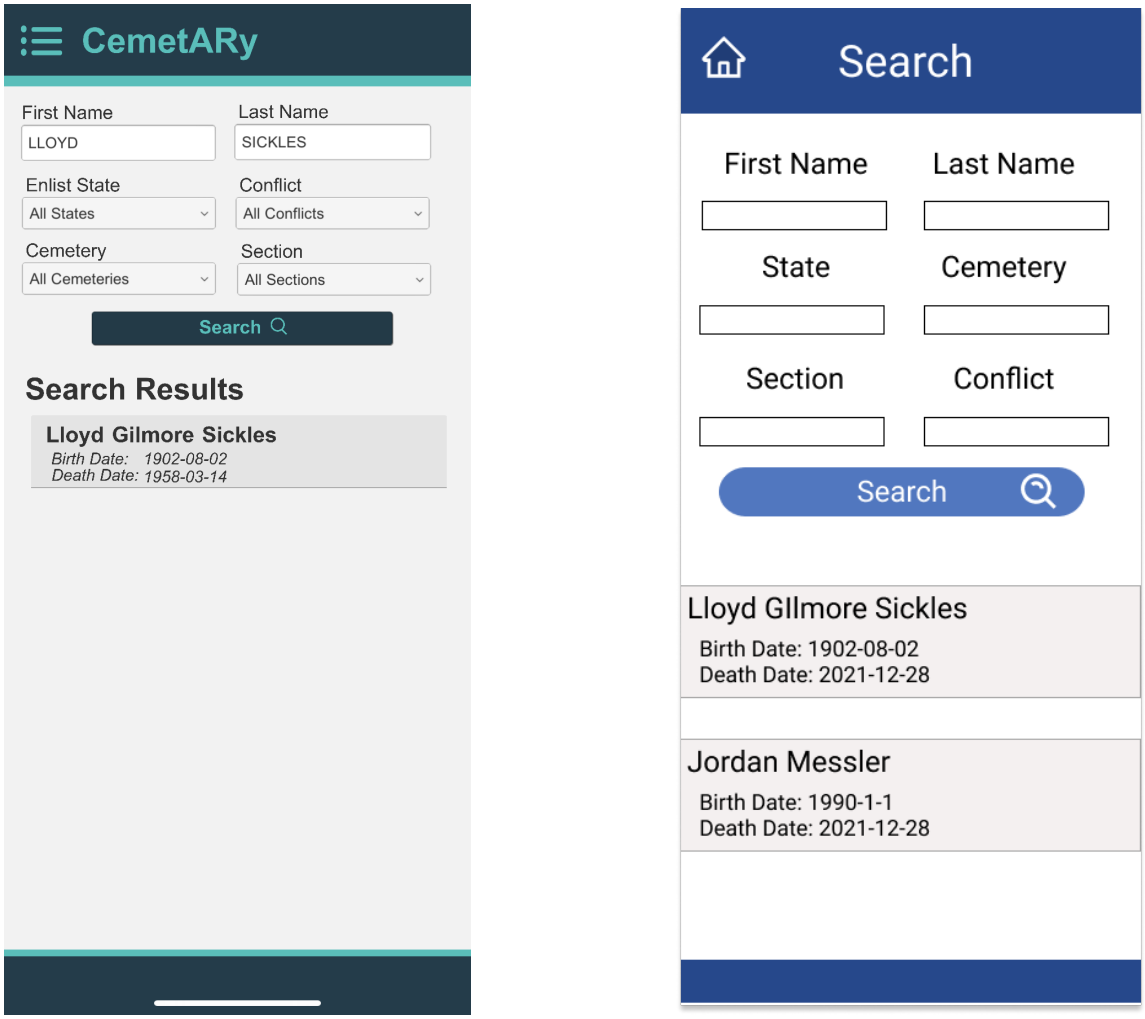


Figure 5.AU. and 5.AV - Search Function Before and After

6. Technology Implementations

6.1 Optical Character Recognition Implementation

For our first implementation of the new optical character recognition system, we decided that the new text detection system developed by the other senior design team will be implemented into the Unity project. This may be difficult, as their optical character recognition system was written in Python. Because Unity uses C#, we may have to rewrite the model to work inside of Unity.

Google Cloud Vision has an API we can use in Unity. This means we should be able to recreate the other senior design team's optical character recognition system inside of Unity.

There are three main components of the other team's senior design project that we would like to carry over into our own. These features are the text detection, the text recognition, and the database searching algorithm for finding the correct veteran.

The first part we need to bring into our current project is the character detection system. The current system requires users to bound where each line of text is on the headstone. Our new system will be able to automatically find where on the picture the text on the headstone is. It does this by detecting the contrast in light between different parts of the image. The new system is able to detect when the text is curved, and can straighten the text to be more easily recognized in the next step.

The next part that we need to bring into our project is the text recognition. This part may not be very different from how it is currently implemented. This part of the program takes the detected text images from the previous part, and figures out what the text on the image actually says.

The third part that we would like to include in our project is the text searching system. In the other senior design team's project, we can take the words we have recognized in the previous part, and find the veteran in the database that these words most likely refers to. This system has allowed the optical character

recognition system to provide data for the correct veteran even if the text was not entirely accurately recognized.

Each of these components need to be included in our project. By doing this, the headstone scanning process will become more accurate and more simple to use.

One issue with this implementation is the lack of online features. Google Vision API cannot be used in an offline environment. Because of this, we may give users the option to select between two optical character recognition systems, with one of them stored locally. However, we will probably not design it this way.

Another potential issue with this implementation is the pricing. Using Google Vision, it would cost us $1.50 for every 1000 uses after the first 1000 uses [26]. Because of this, this could become expensive. This may even be more expensive than implementing the optical character recognition in other ways. However, because this system provides the best results, we will use this system.

There are several reasons we decided to use this system for the optical character recognition. One reason for this is because this system was designed by another senior design group. This team spent time to design this system to specifically solve the problem we are dealing with. Because of this, using this system would greatly improve the performance of our optical character recognition system.

After attempting to implement this system, we decided to implement this system in a different way. This is because Barracuda is limited in what models it is able to use in Unity.

For our second implementation of the new optical character recognition system, we decided to implement it in a different way. To allow for better recognition of curved text, we decided to modify the API call for finding a veteran. In the previous searchforveteran API call, the mobile application would send a list of words with keys associated with them, and the database would return any veterans that matched. We changed this API to allow for substrings of values to count as matches. This allows for more flexibility with the veteran values. Because the optical character recognition system often cuts of the first or last letter with curved text, this new implementation helped solve that issue.

With our new implementation, we also decided to work on making the optical character recognition system easier to use. In the old system, users were required to draw a box around every word on a headstone and identify what part of the veteran that word described. Because we found this system to be more difficult using the search menu, we decided to make the system simpler.

In our new optical character recognition system, we still use Google Vision for detecting and recognizing text. However, now we only have to take a photo of the headstone, and all of the text is detected. We found this solution much simpler to use, as well as faster and less expensive than the previous implementation.

In the previous system, users would draw boxes around each word. Each box the user draws is saved as a separate image, and the text is recognized using Google Vision API. This means that each headstone can have up to six API calls. Our new system only has one API call for each headstone, so our application can detect the text more quickly and cheaper than the previous implementation.

One issue we had to solve with our new implementation was deciding which word on the headstone describes which feature. For example, if a word is found on a headstone, does it describe their first name or last name? The main reason for having users draw boxes around each word was to solve this problem. Because we wanted to only require users to take a photograph of the headstone, we needed to automate this.

After researching the headstones provided to us, we found that headstones consistently have the veteran's first name and last name at the top of the headstone. The headstone may also have their middle name in between the first name and last name. When testing the search menu, we found that these are the most useful fields for finding a specific veteran. Because of this, we implemented a new system so that after the user takes a photo, the text is recognized using the Google Vision API. After the recognized text is received, either the three words closest to the top of the image are used for first name, middle name, and last name, or the first two words are used for the first name and last name. Because not all headstones have a middle name, we included a toggle to allow the user to decide if a headstone has a middle name on it or not.

This new implementation also takes the user to the search menu after the text is recognized. Each field is filled with the values found by the Google Vision API,

and the search section is filled with the veterans that match the search. This is to allow the users to more easily correct mistakes made by the API and search for the veteran in a different way.

6.2. New Search Function Implementation

When searching for a veteran, there are two main things we needed to add. New criteria to be able to search for veterans must be added. Also, on a veteran's page, there needs to be a link to the Veterans Legacy Memorial. Since the base of these features were already implemented in Unity from the previous senior design team's project, these features were relatively simple to add. The search page in the Unity project can have two more criteria added. The user interface may be changed slightly to make room for these new criteria. The criteria must be displayed in a way that makes sense and is easy to read.

The reason we would like to add the new searching criteria is because it would allow for more robust searching. Adding new criteria would allow the users to find the veteran they are looking for more easily, and they can have more confidence that the veteran whose biography they are viewing is actually the veteran they were looking for.

The menu went through a redesign. However, its structure remained largely the same. Because of this, these changes were not difficult to make. These new search and biography features also should have similar implementations to the old search and biography systems. This means that these will hopefully not be hard to implement.

In the old system, when a user viewed a biography for a veteran, they found a button to take them to their Veteran Legacy Memorial. However, the button did not work properly. Also, when the link to the veteran legacy memorial is not found, a red button that says the memorial cannot be found appears in place of the normal button. Another example for a change would be to remove the veteran legacy memorial button when a veteran legacy memorial page is not found. This makes the application easier to understand for the user, by not providing information that is unnecessary. If the button does not appear for a veteran, then the user will know that the veteran legacy memorial page could not be found for the veteran they are searching for.

In our final redesign, the main changes we made are with the sizes of certain elements. This allows for us to include a Veterans Legacy Memorial link for each veteran, as well as allows the application to be run on devices of different screen sizes. In the old version of the application, when a veteran does not have a biography or a headstone image, red text would appear saying that a biography or headstone image was not found. This was removed, because it is not necessary.

6.3. GPS Navigation

CemetARy's Global Positioning System (or GPS) is built on the Mapbox platform. It offers support for Unity, both Android and IOS devices, and is affordable for our budget. Mapbox is free for the first 25,000 monthly users, which is likely more than the necessary demand of most cemeteries. If that is not enough to support a cemetery, it is $4 for every 1000 users, up to the next 100,000 users.

Users can start CemetARy's GPS navigation when they are on a veteran's profile. A veteran's profile can be found by any of the 3 options in the home screen (Figure 3H). Selecting "Find Veteran" will allow users to manually enter information of a veteran to search in the database. Tapping on the desired veteran from the results screen will open up that veteran's profile . Alternatively, a user can choose the "Scan Grave" option on the home screen to use CemetARy's OCR technology to search for the veteran profile (see section: *Optical Character Recognition*).

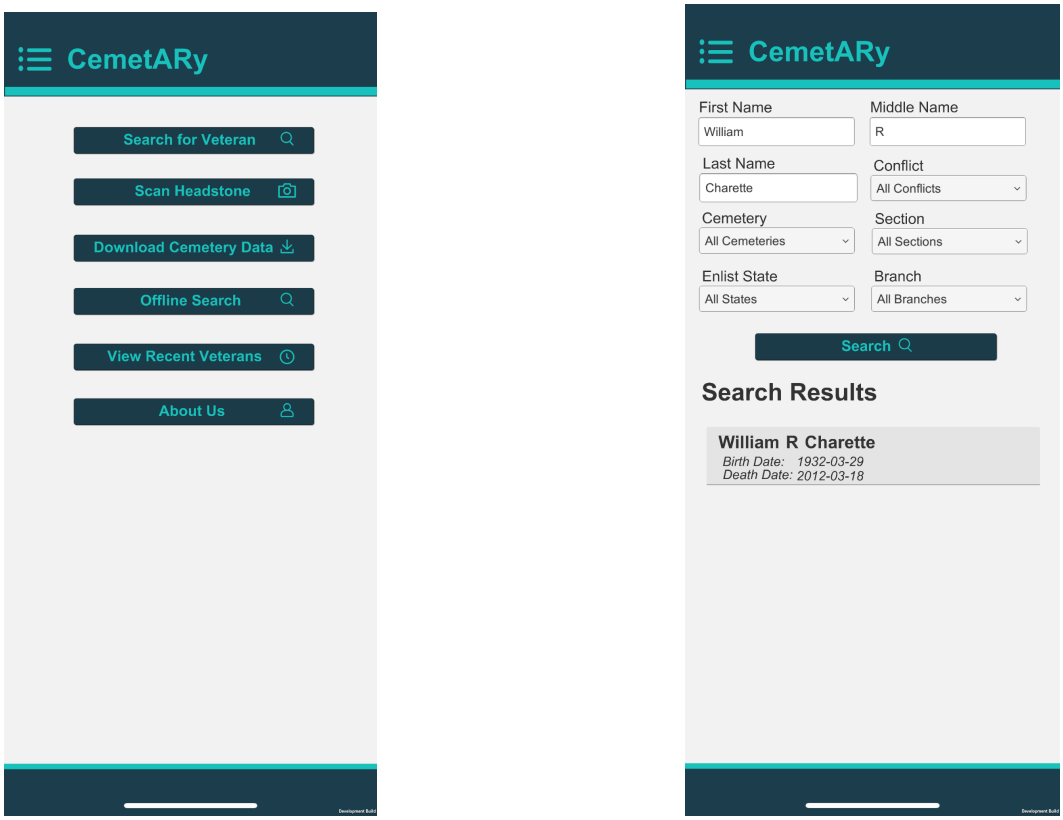


Figure 6.A. and Figure 6.B. - Mobile App Home Page and Search Function

If the user has opened the veteran's profile before, they can access it again from the "Recent" button on the home menu to bring up the recent searches list (Figure 3J). When at the veteran profile, selecting "Navigate to Veteran" will start the GPS navigation, switching over to a 2D top-down view of the area.

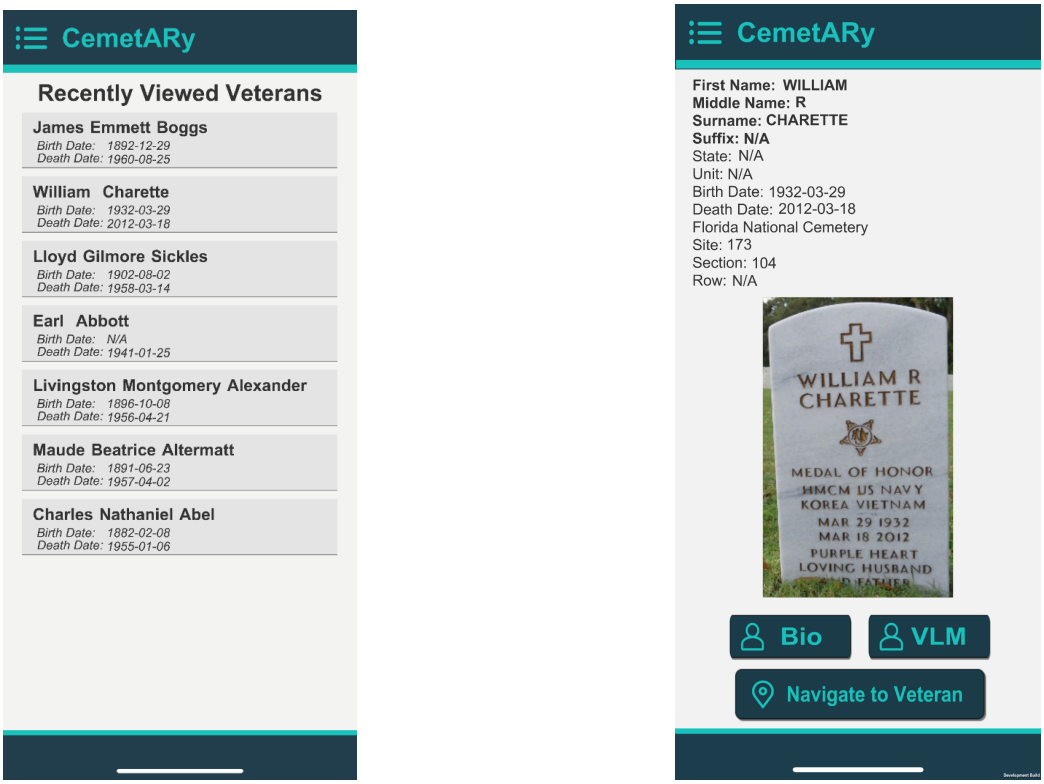


Figure 6.C. and Figure 6.D. - Recently Viewed Veterans pages

CemetARy users can navigate through the 2D map to a selected waypoint. At any time, users can switch to and from augmented reality (AR) navigation, which uses the phone or tablet camera to view the surroundings and project a head-up display (HUD). On the screen, a blue path is charted out from the user's current position to the approximate location of the waypoint.

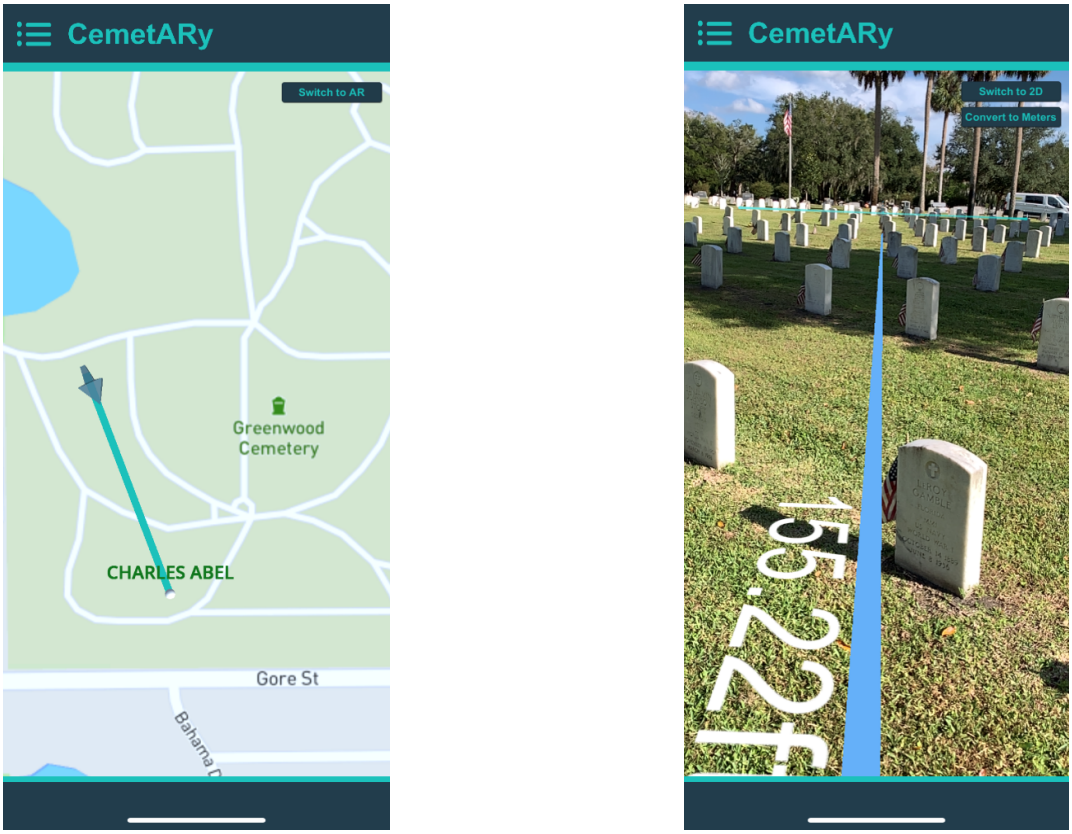
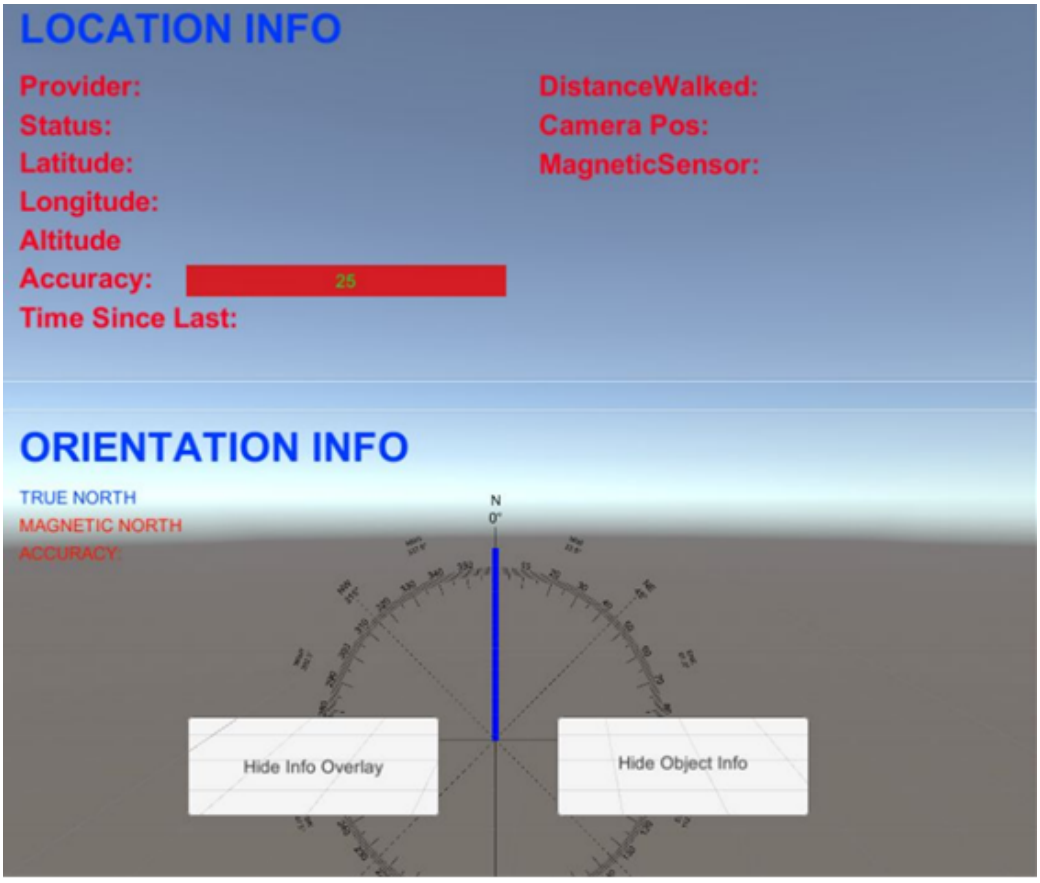


Figure 6.E. and 6.F. - 2D Top-Down and 3D AR Navigation

On the pathing line, the distance from the user to the waypoint is also displayed in feet. A user can also change the distance display to be in meters if desired. Upon arriving at the waypoint, the user will see a 10-meter radius circle that shows the approximate location of the headstone. Currently, this is the best accuracy GPS technology can do. However, the circle is small enough for users to easily search for the desired headstone.

Figure 6.G. and Figure 6.H. - 3D AR distance tracking

AR navigation utilizes the custom Unity AR+GPS library, given by our sponsor, rather than Mapbox. The library enables us to put 3D objects in the AR space, such as the waypoint marker seen in Figures 6G and 6H and the circle bounding the approximate location of the headstone. The library also gives examples on building Unity scenes, which culminated in the current User Interface (UI). On the developmental side, the UI provides more information that is important for testing.



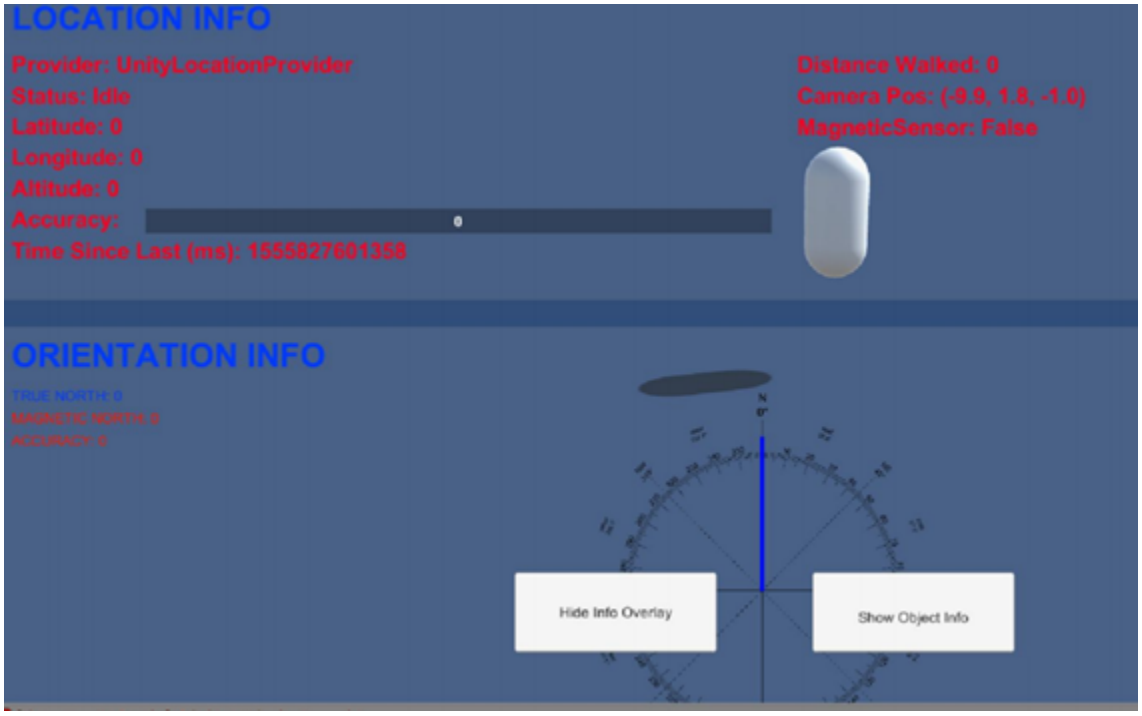


Figure 6.I and 6.J - Developer UI and Orientation

The figures above show the developer UI. In Figure 6.I, you can see a capsule object that has been placed at strict coordinates. While navigating this scene, the capsule will stay in place, much like a real object. This capsule is a basic version of the waypoint marker seen in the user view from Figure 6.G and 6.J. Having these virtual markers serve as strong spatial references when searching for a headstone in the vicinity.

7. Other Mobile Concerns

7.1. Mobile Application Accessibility

Accessibility was an important part of this project for us to consider. This mobile application must be available and work for anyone with an interest in learning more about the history of our veterans. This mobile application must also work on multiple different platforms.

One common issue with accessibility is color blindness. In order to remain accessible to people with colorblindness, we will avoid using colors such as red and green to convey information. All information is attempted to be conveyed in ways besides color.

To make this mobile application more accessible to whoever want to use this application, it needs to function on different platforms. Because the mobile application needs to work on these different platforms, the user interface also has to work well with each platform. For example, this application should work on both iPhones and iPads. These devices, however, have very different screen sizes. Because of this, the application needs to be able to accommodate these differences in screen size.

This application may also help information on our veterans to be even more accessible. Anyone who uses a smartphone for our mobile application to look up information on veterans. They do not even need to be at a cemetery, because the search function, and accessing the information in the database can be done anywhere with a phone. Because of this, our mobile application will make learning about veterans even more accessible. This mobile application will bring accessibility to people who are unable to travel to a cemetery, as well as people who do not have a cemetery near them.

Another feature that we implemented in our mobile application is an offline mode. In this mode, users are able to use the application in an offline capacity. Users can download the data of a specific cemetery to their phone. After doing this, users can access the information on the veterans in a cemetery without having to be connected to the internet. This offline feature helps with visiting cemeteries that have poor internet connection. Because the data is downloaded to their phone, they do not have to connect to the internet, so the poor internet connection in the cemetery does not prevent them from using our application. This feature will help to make our mobile application more accessible to anyone who may use this application in an area with poor internet connection.

7.2. Mobile Application Testing

For this project, we were given the previous team's project to work with.
However, when we make changes to this project, we need to test it to make sure it works, as well as make sure it does not cause new problems due to dependencies with the older project. Because of this, there are many components to this mobile application that must be tested.

When testing this mobile application, another thing we need to consider is that it needs to be tested on multiple different platforms. The mobile application needs to work on iOS devices, such as iPhones and iPads, as well as Android devices.

Since no one on our team has an Android phone, we will be using Android Studio to emulate Android devices. Android Studio is software for building and emulating Android applications on a computer. With Android Studio, we can emulate an Android phone using our computer. Because Unity can export projects to Android, we can then use Android Studio to emulate the application. By doing this, we can get a better idea of how the application works on Android devices, and we can more accurately test the functionality of our mobile application.

The mobile application is made using Unity. This means that we will be using C# for writing the scripts. The C# code will be written using Visual Studio Code. The Unity project can be tested in the Unity environment. We also will use a mobile application called Unity Remote [27]. This application will help us to test our Unity mobile application on our phones without having to deploy the mobile application. This can help the testing to happen more quickly and efficiently.

The current database for the application is hosted on a Heroku server. One of the first things we needed to do was change the server. This needs to be changed so it is hosted on UCF. When this is completed, then we will test if the current system retains all of the functionality with the new server.

7.3. Search Feature Testing

After the database is hosted on the new server, then the search function can be tested. We will test if the application can make queries to the database by searching for veterans and seeing if we are able to get the correct veterans to be returned to us.

After this, there are some features that we need to add to the search menu. For example, we need to add veterans' middle names to be a searchable item. After adding all of the new search terms we would like to add, we will test each one. We will try searching using only the new search terms, as well as different combinations of the old search terms and new search terms. Because the user does not have to enter in all of the information in each search term to make a search, we will test it this way.

Another feature we need to add to the veteran's biography is a link to the Veterans Legacy Memorial. After we have added this button, we can test if it works for the veterans we search for. We need to test if the button works, and also if the button links the user to the correct veteran's biography on the Veterans Legacy Memorial.

Initially these search tests can be done in the Unity editor, but eventually we will also test this on mobile devices. After we are finished testing if the search features work in the Unity editor, we will test them on our mobile devices. We will also test them on different conditions, such as at a cemetery. This is because we would like to test the application in environments where we believe users will actually be using the application.

7.4. Optical Character Recognition Testing

For this project, we will be using another team's optical character recognition system for our mobile application. Implementing this system will take several tests. The first thing we would like to do is get their project running on our computers. We will test the model on the dataset of headstones given to us.

The second step would then be to convert this project into C#, so that it can be used in Unity. We will then need to test this optical character recognition system. After testing the system on the same dataset as the previous optical character recognition system, the systems should have similar results.

After our new optical character recognition system works, we need to implement it into Unity. To test this, we will run the application in Unity and test if the optical character recognition is working. We will test it in Unity using the same dataset of images we used for the previous tests. After we have tested this, then we will know if we have implemented the optical character recognition into Unity correctly.

When we are done testing if we correctly implemented the optical character recognition into Unity, we will need to see if it can scan images from a camera as well. We can test this by using a smartphone to test the mobile application, or by using a webcam while running the application in the Unity editor.

After testing that the application can correctly scan and identify photos from our headstone dataset, we need to make sure it can work in the real world too. For the last test for the optical character recognition, we need to travel to the cemetery to test the application. The mobile application will be on as many different mobile devices as we have available. After this step is complete, we should have a complete and working optical character recognition system on our mobile application.

7.5. Offline Mode Testing

One of the features we added to the mobile application is an offline mode. Cemetery data needs to be able to be downloaded to the mobile device and be accessed when the mobile device is not connected to the internet. There are

several things we need to test before we can be sure that this feature is working as intended.

For the first test of this feature, we needed to test if we can correctly download data from the database to the device. We also needed to be able to access the data from the device. We will test this feature on every device we have available.

When we first implement the search button, it will be an option when a user searches for a veteran in the database. They will use the search feature to find a veteran, and then on the veteran's biography there will be a button to download the cemetery data. This will download all of the data of the cemetery that the veteran is in. After we know that the download button works, we will add a new search menu to search specifically for cemeteries to download. We will add this new feature because it seems more intuitive. We will test the new search menu on multiple devices.

After this is complete, we will test if the application works on devices when they are not connected to the internet. We will need to first download the data, then disable internet access, and then see if the mobile application is still able to access the data that is downloaded to it.

After we have finished that test, then we will need to test if the mobile application can correctly navigate the user to a headstone in offline mode. We will need to visit a cemetery to check if this works correctly. We will use multiple mobile devices to test if it works on all of them and to compare any differences in performance on each device.

7.6. GPS Testing

To test the GPS system of the mobile application, we will first need to test if the search function is working. After we know that the search features are all working, we should also be able to search for a veteran, find them in the database, and click a button to receive navigation to the veteran's headstone. In order to test this, we will be using the Unity editor to search for veterans, and then we will see if we can be navigated to a cemetery, and if the headstone we have been guided to is the correct headstone.

When we are testing the optical character recognition, we eventually needed to go to a cemetery to test the functionality. For this testing, we made a trip out to Greenwood Cemetery, where most of our current data is from. While we were at Greenwood, the GPS navigation was our main focus. The cemetery is quite difficult to navigate on one's own, so it was a wonderful field test.

Conclusions

Sean Grimes

This project has many components to it. There are several different technologies our team needs to use. Because of this, our team needed to learn a lot of different technologies that we are not very familiar with. For our project we need a database, a mobile application, and a web application. All of these things are built using different technology. However, because we are building on another team's project, this workload has been lightened. Through our research process we were able to better understand the technology we will be using, as well as better understand the previous team's project that we will be working on.

Our sponsor for this project has been very helpful during the development of this project. Dr. Giroux has had frequent meetings with us and has helped to make the requirements for this project very clear. She has also been available and helpful whenever we had questions about our project or the previous team's implementation. Because of our meetings, we were able to show the changes we made between meetings, which was helpful to progress.

I have some experience using Unity before working on this project, which has helped to make some of the learning process easier for me. However, even with previous knowledge, I still had to do a lot of research, and I had to gain a better understanding of Unity. Learning the previous team's project, and implementing changes was a rewarding experience for me. I had not used Unity that much before for anything other than games. I also was able to learn a lot about computer vision because of the optical character recognition. We initially planned to implement another team's OCR system, but we ultimately went with rewriting the current system to only require one image. Working on this project has been a

valuable experience for me, and I learned many things about both programming as well as project and team management.

Tevfik Koyun

I have been very motivated while working on this project. Working on a real project has been very exciting. I spent most of the time this semester learning skills and trying to understand the work of the previous team. The project is currently working with a few small mistakes. I think that working on a real world project before graduation is amazing for us.

Also this project is about veterans, which is something that feels particularly honorable. We are working for their memories and their families. Our fallen heroes and their memories deserve as much appreciation as possible for what they have done and will do for us. We hope that we can interest new generations with this application in what our veterans have done.

This semester, I worked on database and backend development. I have never worked this deeply on the backend and database before. At the beginning, it seemed very easy, however when I started to understand the previous team's work, I was awed by how well they had constructed the project. They made approximately 20 API endpoints. Next semester, I will be working on the database as well as constructing and quality checking API endpoints.

This semester, we made changes to the server as our sponsor requested. For the next semester, my concern is that these API endpoints will not work for the database in the new server. As a team, we may need to change many features as we continue working.

This project was my first choice when given the selection of project pitches. As the first semester ends, I am very happy that I am working on this project. As a team we always help and respect each other. Finally, I am glad to work with Dr. Giroux and Connie Harper. They attend and host meetings regularly and are very helpful in guiding us as we work on this project.

Jordan Messler

This project involved a lot of new technologies our team has never used, so it was quite intimidating for us. I was taken aback by how many learning components were necessary to even understand what the previous team accomplished.

Perhaps the more shocking thing, however, was not the learning material. I could never have anticipated the resources we received to help us. Dr. Giroux and Connie Harper were always available for help when we needed it -- often meeting with us outside our scheduled meeting times. Dr. Giroux also provided us with plenty of organized reference material for the project, including some of the raw data that went into the database. On top of that, we were put into contact with another group that focused on one of the most technically difficult portions of our project. We often spoke with Trevor from the team, who pointed us to various online resources on how to build OCR solutions, and took us through his own.

As for the rest of my team, I've been impressed. Despite difficulties, we were diligent in our meetings and work. We have been communicating well, and everyone makes sure they participate. The teamwork, combined with the resources and assistance we received, allowed us to deliver on our project goals.

Kyle Wu

It's been an honor and a pleasure to work on this project. I've learned firsthand what it's like to be on a development team and also the amount of work that I need to put forth in order to be a good team member. There was a lot to learn and procrastinating really wasn't a possibility for our team; we needed every second that we had in order to make things work.

Learning React and then immediately putting that knowledge to use was an incredible experience, as was working with my team. Perhaps most importantly, the project we worked together on was very meaningful. Many thanks to Dr. Giroux and Connie Harper for very patiently working with us as we tried our best to figure out the ins and outs of the project.

9. References

1. Hale, Coda. “How To Safely Store A Password.” How To Safely Store A Password , 2018, codahale.com/how-to-safely-store-a-password/.
2. D“React: The Virtual DOM.” *Codecademy*, Codeacademy, 2021, www.codecademy.com/articles/react-virtual-dom.
3. Wikimedia Foundation. (2021, August 2). *Coupling (computer programming)*. Wikipedia. https://en.wikipedia.org/wiki/Coupling_(computer_programming).
4. “Pete Hunt: React: Rethinking best practices -- JSConf EU” *YouTube*, uploaded by JSConf, 13 October 2013, https://www.youtube.com/watch?v=x7cQ3mrcKaY
5. “Angular (Web Framework).” *Wikipedia*, Wikimedia Foundation, 30 July 2021, en.wikipedia.org/wiki/Angular_(web_framework). https://en.wikipedia.org/wiki/Angular_(web_framework)
6. “Vue.js.” *Wikipedia*, Wikimedia Foundation, 18 July 2021, en.wikipedia.org/wiki/Vue.js https://en.wikipedia.org/wiki/Vue.js
7. Harkushko, Liliia. “Angular: Best Use Cases and Reasons to Opt for This Tool.” *When and Why Angular Is a Good Technical Solution for Your Project*, yalantis.com/blog/when-to-use-angular/. https://yalantis.com/blog/when-to-use-angular/
8. Kodytechnolab.com. “When and Why You Should Use Angular for Your next Project?” *Kody Technolab*, 22 May 2020, kodytechnolab.com/why-use-angular.
9. “Comparison with Other Frameworks - Vue.js.” - *Vue.js*, vuejs.org/v2/guide/comparison.html.
10. “Beautiful Native Apps in Record Time.” *Flutter*, Google LLC, flutter.dev/.
11. “mapbox_gl | Flutter Package.” Dart Packages, 12 Apr. 2021, pub.dev/packages/mapbox_gl.

12. “arcore_flutter_plugin | Flutter Package.” Dart Packages, Gdifrancesco.dev, 15 May 2021, pub.dev/packages/arcore_flutter_plugin.
13. “arkit_plugin | Flutter Package.” *Dart Packages*, 6 June 2021, pub.dev/packages/arkit_plugin.
14. “Opencv | Flutter Package.” *Dart Packages*, 11 Nov. 2020, pub.dev/packages/opencv.
15. “Using OpenCV in Unity.” *Stereolabs*, Stereolabs Inc., 2021, www.stereolabs.com/docs/unity/using-opencv-with-unity/.
16. “Maps SDK for React Native | Help | Mapbox.” *Mapbox*, docs.mapbox.com/help/glossary/maps-sdk-for-react-native/.
17. “UIWebView.” Apple Developer Documentation, Apple Inc., 2020, developer.apple.com/documentation/uikit/uiwebview.
18. Media, Viro. “Viro Is Going Open Source!” Medium, VR & AR App Development Blog - Viro Media, 16 Oct. 2019, blog.viromedia.com/viro-is-going-open-source-be9da0b43328.
19. “Introduction to Barracuda | Barracuda | 1.0.4.” Barracuda | 1.0.4, Unity Technologies, 20 Oct. 2020, docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html.
20. Morales, Fausto. *Using Pretrained Models - keras_ocr Documentation*, 2019, keras-ocr.readthedocs.io/en/latest/examples/using_pretrained_models.html.
21. “Colaboratory.” Google, Google, research.google.com/colaboratory/faq.html.
22. “Tesseract OCR - Opensource.google.” Opensource.google, Google LLC, opensource.google/projects/tesseract.
23. “Vision AI | Derive Image Insights via ML | Cloud Vision API.” *Google*, Google LLC, cloud.google.com/vision#section-8.
24. “Using OpenCV in Unity.” *Stereolabs*, Stereolabs Inc., 2021, www.stereolabs.com/docs/unity/using-opencv-with-unity/.

25. National Cemetery Administration. (n.d.). *Veterans legacy MEMORIAL: U.S. Department of Veterans Affairs*. VLM. https://www.vlm.cem.va.gov/.

26. “Pricing | Cloud Vision API | Google Cloud.” *Google*, Google LLC, cloud.google.com/vision/pricing.

27. “Technologies, U. (n.d.). *Unity Remote*. Unity Documentation. https://docs.unity3d.com/Manual/UnityRemote5.html.

10. Appendix

Software Used

Figma : https://www.figma.com/

Unity: https://unity.com/

Trello: https://trello.com/

TeamGantt: https://www.teamgantt.com/

Android Studio: https://developer.android.com/studio

Visual Studio Code: https://code.visualstudio.com/

Google Colab: https://research.google.com/colaboratory/

Google Docs: https://www.google.com/docs/about/

Discord: https://discord.com/

Github : https://github.com/

Cisco: https://www.cisco.com/

Mapbox: https://www.mapbox.com/

Discord: https://discord.com/

GitHub: https://github.com/

Zoom: https://zoom.us/

Winscp: https://winscp.net/eng/index.php