UNIVERSITY OF CENTRAL FLORIDA

# The Gravestone Project

June 20th, 2021


**Group 3 (TGP)**

*Shelby Menown*

*Alex Ravelo*

*Jayden Sipe*

*Timothy Ziemathis*


**Sponsors**

Dr. Amy Larner Giroux, University of Central Florida, Associate Director of the Center for Humanities and Digital Research

Dr. Emily B. Stanback, University of Southern Mississippi, Associate Professor

# Contents

# 1 Executive Summary

Artifacts left behind by or in memorial of the dead are a significant part of societies and cultures around the world. Many groups have put the effort forth to try and create digital archives of various types of artifacts, often focusing on the archival of gravestones or obituaries. Saving the information of artifacts relating to the dead in a digital location has become incredibly important for scholars, teachers, students, writers, and many others to reasonably study and analyze such data. Currently, these groups are in need of a centralized location that can provide these tools.

Our project aims to be this tool; a tool to serve as a singular source for the collection and analysis of all types of artifacts of the dead--a concept approached many times, but lacking accessible or upkept digital solutions. This has led to mass amounts of culturally significant data remaining out of reach to scholars and non-scholars alike. The importance of digitally memorializing artifacts of the dead is not lost on any member of our group—we all share a great admiration towards the effort of collecting, preserving, and analyzing this data.

The Gravestone Project (TGP) is a website that serves as a digital humanities collective that brings together scholars, taphophiles, students, writers, teachers, and others interested in history, literature, and the arts, to think about the various ways that people memorialize the dead. The aim of this project is to help scholars collect death related artifacts by expanding the functionality of the existing website. This expansion entails providing an expansive search function for a database of artifacts, a way to request additions to the archive, transcribing artifact data, performing textual analysis of the collected data, and mapping the locations of queried data.

This document will serve to detail both our research towards adding the archival and search extension to the existing TGP website, and our design plans to implement it. We will be developing the website with a MERN stack, subbing MongoDB for MySQL, and hosting off of The Gravestone Project's CHDR server. Upon completion of the website, standard users will be able to perform searches for artifacts, utilize content filters, and map results. Our hope is that, with these additions, the website can become a centralized location for the archival of death-related artifacts, and that it may help to grow the TGP community.

# 2 Project Overview

## 2.1 Project Description

Throughout the times, many people and cultures have honored, recorded, and even celebrated the lives of the recently deceased, and have constructed various gravestones and artifacts to mark and memorialize close friends and loved ones. The Gravestone Project (TGP) is a digital humanities collective that seeks to center the methodologies of the humanities, histories, and arts to investigate aspects of 17th to 19th century cemeteries and memorials. The main purpose is to start projects that allow academics and the general public to collaborate in the creation of knowledge about the history of those cultures and these individuals' lives.

In order to achieve this, we have been given an existing website (http://thegravestoneproject.com/) that contains the basic ideals of the project, such as a list of already documented artifacts. Our job is to take this existing website and expand its capabilities. We will add be adding a robust search functionality for artifacts, with users being able to map out where artifacts are through a service such as Leaflet to discover trends between multiple artifacts, translations of any foreign language transcriptions on the artifacts, and an in-depth tagging system to group similar artifacts together.

One new feature we will be adding to the site will be to allow users to submit any artifacts that are not on the site already, with a form hosted on Google Forms. This simple feature allows this project to grow from just a small personal project to a community effort. Anyone who is generous enough to contribute, whether it be someone who wants to preserve the headstone of a loved one or a professional photographer who has taken higher quality photographs of certain artifacts. To further push the project as a community effort, there is a Twitter account tied to the project, and users may also interact with the social media manager to provide artifacts or ask questions pertaining to the site or project. As the community grows larger, more gravestones and artifacts are documented and preserved, and as the size of the data we are preserving grows larger, it will attract more people to the site and the cause, thus growing the community that the site is made to nurture. This creates a loop that will cause a healthy growth of the site.

We will be using a one of UCF's Center for Humanities and Digital Research servers, a CHDR server, to store data in the database, in which site administrators will be able to interact with. Administrators will have various tools and functions that basic users will not have access to. Through these tools, admins will be able to add artifacts to the website, edit the information of an existing artifacts, edit artifact-type (gravestone, obituary, etc.) and relational-type (mother, sister, wife) tags, ban or delete malicious users, and promote or demote users between administrative and standard account types.

Ultimately, our work is a giant first step to the future of The Gravestone Project, and our sponsor has expressed interest in expanding on the greater project even further through both concurrent and future projects, so we have taken careful steps to construct and document the website in a way that allows room for growth and expansion. This allows future web developers, be they professionals or students, to improve on the site or add features with ease after our departure.

## 2.2 Statements of Motivation

Below are personal statements from each member of the team regarding their motivation to join the project, and their desire to ensure it comes out right.

### 2.2.1 Shelby Menown

This project has a personal significance to me, as seeking out information of those who have passed has been a theme within the family. However, any time my family members have tried to unearth this information, the process has been long, tedious, and stressful. I'm excited at the prospect of providing a service that consolidates this search for surviving loved ones, for scholars, and for the simply curious. My goal, and likely the entire group's goal, is to ensure this service is easy to use, access, and maintain after our work comes to an end. Our contributions to The Gravestone project should be something we can take pride in.

I would also enjoy making it to the stretch goals; I'm something of a statistics nerd, and think the ability to easily analyze this kind of data would be fascinating. But even with the drive to go above and beyond, the first and most important goal is to give 100% effort towards the initial requirements.

### 2.2.2 Alex Ravelo

This project seems very unique and interesting. I believe that preservation is very important for a variety of reasons, such as knowledge, cultural significance, and environmental stability in certain cases.

This project specifically would preserve memories of those no longer with us through photos of their graves and artifacts as well as their obituaries. This, of course, would be useful for research purposes, but I believe this app is more useful to preserve memories of family members and loved ones, so they could reminisce if they are unable to do so physically. My hope is that we could make an important tool that would still be helpful past our own lifetimes.

### 2.2.3 Jayden Sipe

The Gravestone Project seems quite interesting, in the sense we are dealing with such an unusual subject. We are dealing with artifacts that can be potentially thousands of years old and could have so much important history embalmed in them. When I think of artifact collection and storage, my mind wanders to an old-timey historian looking through scrolls and such that includes information about the artifacts on them. This may not be necessarily true nowadays, but having the opportunity to modernize the entire process of artifact collection, storage, and analysis is pretty special. My hope is that we can deliver a fully functional product that meets and exceeds every standard our sponsor puts upon us, and have a great time doing it.

The goal for this project is to create a back-end capable of reading and storing artifact data that can be used to query data such as an artifact’s place where the item resides, owner’s place of birth, place of death, etc. Some examples of artifacts are gravestones, obituaries, samplers, oral histories and literary works.

### 2.2.4 Timothy Ziemathis

I find the Gravestone Project interesting because it's preserving the history of others and creating a tool that could one day help preserve our own information. I look forward to helping with this tool that can help people who may be doing research on their own family and store information such as pictures and locations especially for those who may not live near the grave. I’m also excited because the skills I want to learn the most are web development and database interaction which

are two of the requirements for this project. Getting more experience in a real group project is great!

For goals I would love to meet some of the stretch goals, especially the interface with Voyant Tools because I find data visualization interesting. My goals for this group and project are to go past the standards that were set before us, to have little to no crunch time in the end, and have fun making it with the group.

## 2.3 Initial Ideations

### 2.3.1 Shelby Menown

- With TGP's concerns for ensuring that all data is presented in a respectful and appropriate way, users should be able to submit reports to inform admins when they find a misrepresented/inappropriate listing of an artifact, and should be able to recommend changes to said artifact.
- Implement user accounts as a way to allow user submission directly through the website, instead of through Google Forms. User submission would still need TGP admin approval, but would consolidate the process to the site.
- Ensure the website achieves its core purpose (as a medium for the storage and analysis of death-related artifacts) by taking into consideration the features and implementations of websites that aim for the same or similar goals (ex: obituary websites, grave marker websites).
- Ensure the website is easily accessible and upholds modern design standards by drawing inspiration from the features of other highly maintained websites, both related and unrelated to the project objective. (Ex: Google's mapping feature, Amazon's search filters, FindAGrave's advanced search options).

### 2.3.2 Alex Ravelo

- Users should be able to query results to find artifacts in many different ways (place of birth, cause of death, etc.)

- Make sure the database will be able to receive many different forms of artifact submission, such as: only having one picture, missing information, extra notes for the artifact, etc.
- Allow user submission by Google Forms or by implementing a way of reading a spreadsheet. Our sponsor also mentioned that they are working on something to allow users to send an artifact described by JSON.
- Figure out how to authorize user submissions so users cannot submit mischievous data. Perhaps the admins of the website will authorize each user submission, or only allow admins to submit artifact information once filled out on Google Forms / sent by JSON.

### 2.3.3 Jayden Sipe

- Create a website using whatever backend language the group is most comfortable with and is appropriate for the project, for myself that is PHP.
- The main table of the database can be organized to include the information from Dublin Core for all of the artifacts.
- Coordinates can be stored for each artifact and then different queries can be used with the Leaflet JavaScript library to map locations.
- Users will submit artifacts through google forms and admins will manually add them to the database. Another option would be to allow users to submit multiple entries at one time through data in an excel sheet.
- For images the database will communicate with a file storage system, either local or cloud depending on what the sponsor wants. There can be one directory for user submissions and then a separate directory for every organization that enters data through Zooniverse.

### 2.3.4 Timothy Ziemathis

- Move away from the current website model. From digging through the source code, it seems that each grave is hardcoded into the index file, which

may get messy as more graves get added. The new website model would pull from the database rather than have elements be hardcoded.

- Making sure that the gravestone/artifact submission process is as accessible as possible to avoid complications or frustrations. Have a moderation system in place to reject spam, jokes, or empty submissions.
- Making sure each entry on the site has enough relevant information, and to make sure it is formatted correctly. Plenty of entries on the current site have missing information that is causing awkward empty space, and images that are shown in entries do not have proper formatting, causing the page to look messy.
- The site itself should be heavily accessible. As a historical and cultural tool, being able to relay information in an easy manner that is not intrusive will help it stand the test of time.

## 2.4 Goals and Objectives

- Provide a service for scholars and other interested parties to easily search, view, map, and analyze data for various artifacts of the dead.
- Provide a tool for The Gravestone Project (TGP) members to collect data on death related artifacts.
- Provide website tools that will allow TGP members to easily maintain the site, upload artifacts, and manage users.
- Increase awareness of The Gravestone Project through the added functionality to the website.
- Help to create and expand the community surrounding The Gravestone Project.
- Help to expand the collection of death related artifacts in a singular service.

## 2.5 Constraints

- The wireframe of the product must be completed solely by the members of the group before the end of Senior Design 1.
- The website extension must adhere to the stylistic choices present on the current TGP website.
- The database must be set up solely by the members of the group before the beginning of Senior Design 2.
- Project must be completed solely by the members of the group before the Senior Design Showcase at the end of the Fall 2021 semester.
- Project has no allotted funds; members are to seek free solutions to design problems before considering paid solutions, as costs will be out-of-pocket.

## 2.6 Target Audience

While the main purpose of the site is to document artifacts, there are certain distinct types of users that can be grouped together, such as:

- Scholars
- Taphophiles
- Students
- Writers
- Teachers
- Those with an interest in the arts
- Those with an interest in literature
- Those with an interest in history
- Those with an interest in statistics
- Those who want to search for or preserve the graves of loved ones

## 2.7 Broader Impacts

Our project involves gathering information on gravestones from the 17th to 19th century and making this information more accessible to the public. The Gravestone Project is an important project for understanding and visualizing history. Having birth, death, and life records available publicly with search filters and the ability to map out different artifacts could result in new discoveries. For instance, it could give researchers a better idea of where to look for other graves or physical artifacts of a certain individual or a time period.

The gravestones in the project are not always beautiful intricate designs. Sometimes they are graves of individuals who could not afford a proper gravestone. The project will shine a light on the lives and deaths of people in the 17th to 19th century. This includes your average worker to wealthy individuals to individuals who were enslaved or had physical or mental disabilities and resided in asylums. It is important to keep this history documented so these individuals are not forgotten.

## 2.8 Legal, Ethical, and Privacy Issues

The Gravestone Project is unsurprisingly related to the artifacts of individuals who have passed away. It is important to respect the dead, and treat their memorialization with care. We must find a balance of archiving history, while still respecting the deceased, their families, and people with similar situations or disabilities if applicable. It is important to be careful and respectful regarding what artifacts and personal information are displayed on the website.

Some of the artifacts within the Gravestone Project include individuals who were enslaved, colonized, or lived in asylums. In these situations, it is important to reach out to the relevant communities to collaborate and get input to determine ethical concerns of that marginalized group and appropriately represent them. Currently, the Gravestone Project is limited to the United States, United Kingdom, and the Caribbean - specifically Jamaica - since they have connections to community members in these areas. The Gravestone Project does not want to expand to other locations without having a community outreach to make sure they are collecting and presenting this data in a respectful and appropriate way.

The data on the site comes from publicly accessible locations--not private gravesites or artifact resting sites. Care should be taken when accepting artifact submissions such that none of are from private locations, or have inaccurate information.

Most photographs on the site were taken by an individual working for the Gravestone Project; however, if other photos are submitted, proper permission and credit the photographer is a must. Submissions are allowed on Ttwitter as well and we will collect the person’s Twitter handle should they specify that they want credit.

# 3 Specifications and Requirements

Listed below are the requirements and stretch goals for The Gravestone Project website. The Front-End requirements can also be used to describe the requirements of the overall system.

## 3.1 Front-End

### 3.1.1 Requirements

- Allow all users to create an account, edit/recover their account, log in and log out. Users will need to supply a valid email and a strong password.
- Allow admins to add singular artifacts to the database by manually filling out the data on a form.
- Allow admins to add/import collections of data into the database by uploading a CSV file.
- Allow basic users to request the upload, edit, or deletion of artifacts on the website. Users will be redirected to a Google Form where they can fill out their request. Information about the artifact to be collected may include the artifact title, given names, person type, surname, birthdate, death date, birth location, death location, cause of death, images, transcript, and potentially transcript translation. Regarding the grave itself, information to be collected includes the grave's country, state/providence, site name, longitude, and latitude. Information can be uploaded via CSV file.
- Map the website such that users can select links that will redirect them to the appropriate pages on the main TGP website: http://thegravestoneproject.com/.
- Allow all users (basic and admin) to view a list of all artifacts. Users will then be able to click on an artifact to pull up all associated information.
- Allow all users to search and filter for all artifacts in the database. Users will be able to open up a map and view results shown as pins placed at their

respective locations. Users will be able to move around the map to look at all results.

### 3.1.2 Stretch Goals

- Allow admins to view website usage analytics through a restricted (admin-only) webpage. Information may include search data, artifact viewing data, and more.
- Allow admins to export metadata from the website.
- Allow all users to visualize trends in transcripts and other texts on the website through an interface with Voyant-Tools.
- Ensure the web application works as intended on mobile/tablet devices.

## 3.2 Back-End

### 3.2.1 Requirements

- Provide endpoints for logging in and signing up.
- Provide an endpoint for temporary account lockout upon three incorrect password attempts in quick succession.
- Provide endpoints for all functionality that triggers an email - email verification, password reset, and notice of locked account
- Provide an endpoint for searching artifacts.
- Provide an endpoint for getting all admins and all basic users.
- Provide endpoints for approved admin to promote, demote, or delete other user accounts.
- Provide endpoints for adding, editing, and deleting tags.
- Provide endpoints for adding, editing, and deleting artifacts.
- Provide an endpoint for uploading artifacts from a CSV file.
- Ensure the proper storage of images in the CHDR server and record the file location in the database. Images uploaded during a CSV upload will exist in

a subfolder of the main image folder, named after the artifact collection. Images uploaded to singular artifacts will exist in the main image folder.

- Use publicly available APIs for better functionality - Leaflet for mapping, and Twilio Sendgrid for sending emails
- Database is to be on the existing TGP CHDR server, and the database must be a MySQL database. Connecting to the database is secure and requires using a VPN.

### 3.2.2 Stretch Goals

- Save the most common searches to suggest to users. See if we can implement this by keeping track of most common searches through Google analytics or save searches when signed into account.
- Provide the user an option to select a random artifact to read about. This will randomly return an artifact in the database.
- Use Twitter API to implement a live feed of TGP's Twitter account on a page.
- Add in occupation for each person in the database.
- Have a way to distinguish uploaders as twitter users so we know their username is their twitter handle.

# 4 Division of Labor

## 4.1 Individual Responsibilities

This section provides a brief overview of what each member of the team is responsible for during the development of the project. Several larger responsibilities may be split between team members, and will be listed as identical bullet points under each applicable member. Members are listed in alphabetical order of their last names.

### 4.1.1 Shelby Menown

Shelby is the project lead and front-end development lead. Her responsibilities throughout the course of the project include:

- Project management: Track all tasks in all components, adjust due dates and ensure that due dates for tasks are being met, provide assistance to help keep components on time, facilitate communications between sponsor and other relevant parties.
- Development of website design via Figma wireframe/prototype.
- Determining the structure and design of web pages.
- Ensuring user experience determines design choices.
- Developing or overseeing the development of features to enhance the user experience.
- Striking a balance between functional and aesthetic design.
- Utilize React.js primarily to envision ideas.
- Building reusable code for future use.
- Optimizing web pages for maximum speed and scalability.
- Utilizing a variety of markup languages to write web pages.

### 4.1.2 Alex Ravelo

Alex is a database manager and a member of the API development team. His responsibilities throughout the project include:

- Developing certain API endpoints.
- Enhancing the scalability and performance of API endpoints using REST API.
- Enhancing the scalability and performance of existing database architecture.
- Developing database structures and features according to project needs.
- Performing database maintenance, migration and best practices in database management with MySQL.

### 4.1.3 Jayden Sipe

Jayden is a member of the frontend development team and a flex member of the API development team, alongside working to ensure proper server communications and connectivity. His responsibilities throughout the project include:

- Ensuring user experience determines design choices.
- Developing features to enhance the user experience.
- Utilize React.js primarily to envision ideas.
- Building reusable code for future use.
- Optimizing web pages for maximum speed and scalability.
- Utilizing a variety of markup languages to write web pages.
- Aiding in the adjustment and development of API endpoints.
- Testing features and documenting bugs.
- Protecting data by developing data security and restoration policies, procedures, and controls.
- Developing procedures to ensure data integrity and quality.

### 4.1.4 Timothy Ziemathis

Tim is the API development lead and a database manager. His responsibilities include:

- Developing or overseeing the development of all API needed for the website.
- Enhancing the scalability and performance of API endpoints using REST API.
- Developing database structures and features according to project needs.
- Developing procedures to ensure data integrity and quality.
- Performing database maintenance, migration and best practices in database management with MySQL.

## 4.2 Gantt Chart

As shown in Section 4.1, each team member has their individual responsibilities. These responsibilities are dependent on one another for the development of the project, as certain goals that need to be met by certain members will be contingent on the completion of a prior goal by another member. The following is a Gantt Chart illustrating these dependencies, split into each Sprint. Sprint 1 targets Accounts and Searching, Sprint 2 targets Administrator Functionalities, and Sprint 3 is for Senior Design Showcase Preparation

The tasks are broken down by color, rather than by name, to represent teams. Red tasks are for the Front-End team: Shelby Menown and Jayden Sipe. Green Tasks are for the Back End team: Timothy Ziemathis, and Alex Ravelo. Yellow tasks are for a subset from both teams. Blue tasks are tasks that do not lean towards either team’s specialties, and are be tackled either by the entire group, or by a subset selected as the task’s beginning approaches.

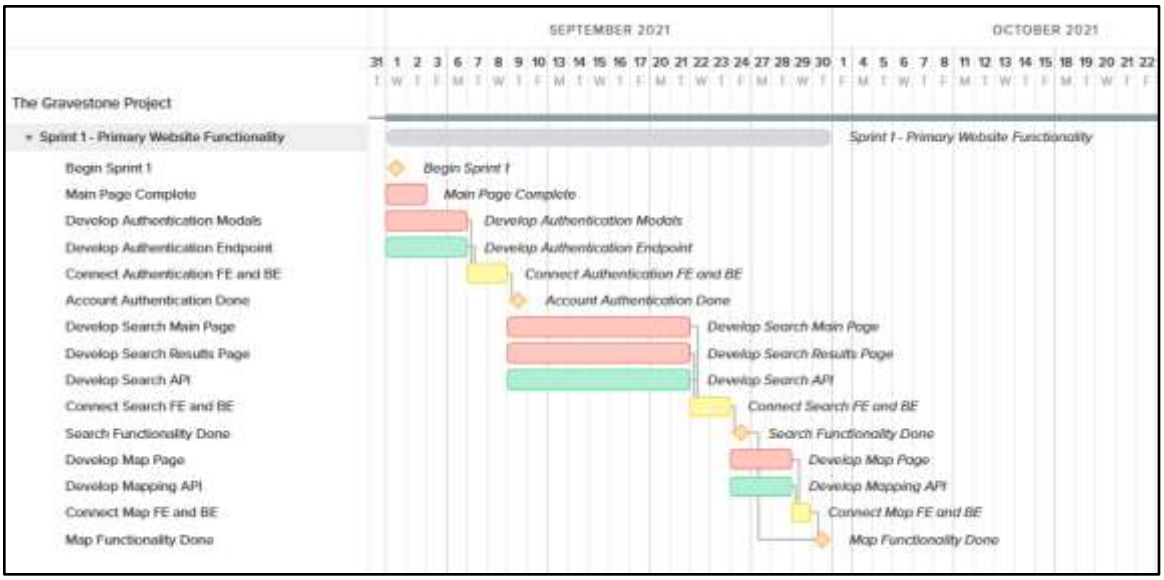


Figure 1: The Gravestone Project (TGP) Gantt Chart (Sprint 1)

Sprint 1, pictured in Figure 1, will span over the month of September, and focus on implementing login, signup, email verification, password recovery, as well as the entirety of the search functionality, both in list form and map form. This month's goals may be the most ambitious (with the given timeline), but our hope is to work hard to meet these goals, allowing us breathing room for the remainder of development.

4.1

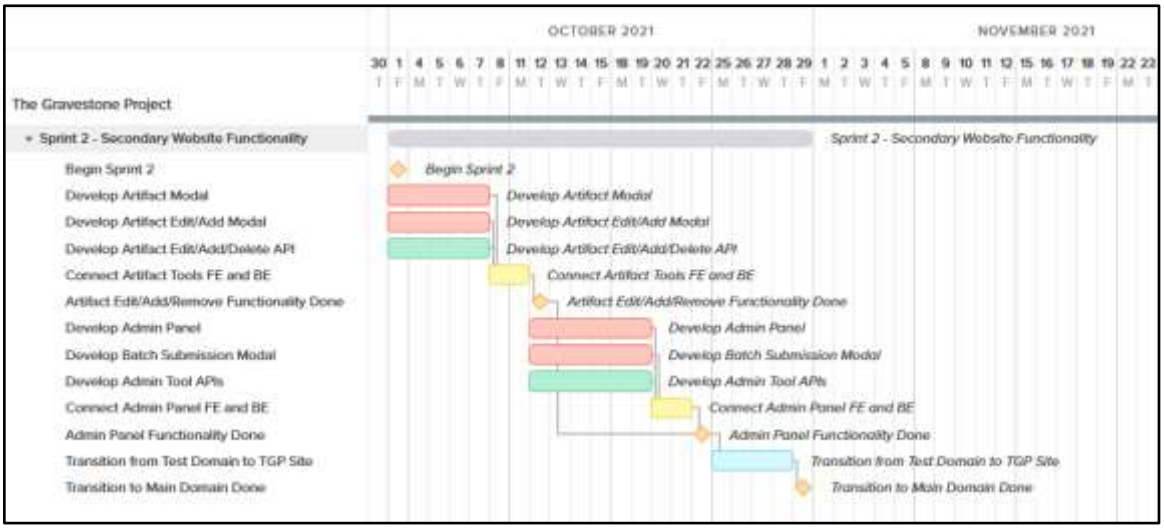


Figure 2: The Gravestone Project (TGP) Gantt Chart (Sprint 2)

Sprint 2, pictured in Figure 2, will span over the month of October, and focus on artifact viewing and administrative tools like user promotion/demotion, tag editing, and general system maintenance. This month's tasks are given slightly more

development time than last month's with the consideration that we may need to extend Sprint 1 into early October (giving us the space to decrease Sprint 2's time allotments).

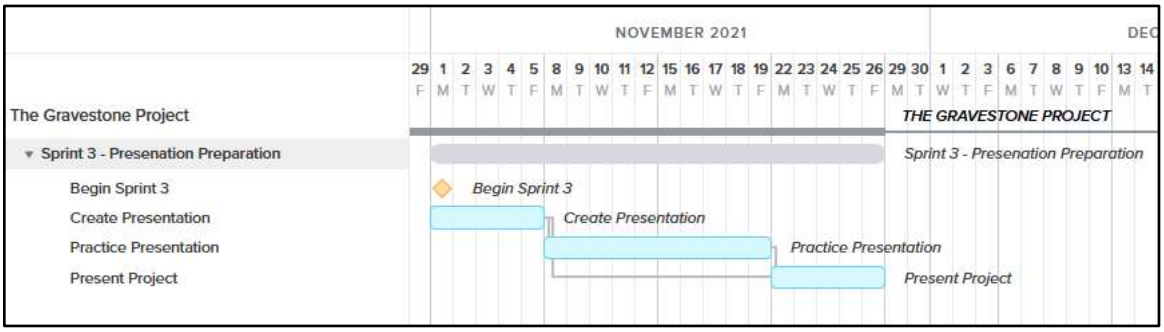


Figure 3: The Gravestone Project (TGP) Gantt Chart (Sprint 3)

Sprint 3, pictured in Figure 3, will span over the month of November, and focuses solely on preparing the already-complete project for the Senior Design 2 presentations. A single week will be spent preparing the presentation, and all remaining time will be devoted to practicing the presentation. "Present Project" is presented as a task, rather than a milestone, because we have no specific dates for the Senior Design showcase, other than that the showcase is held in late November.

# 5 Research and Investigations

In this section is the research we conducted during this project. Research includes what programming languages or stack we use for the web app and information about third party services used in the application.

## 5.1 Stack Components

### 5.1.1 MySQL

Web applications can be made with practically any technology, so figuring out which technology to write with can be a challenge. We were restricted to using a MySQL database because one was already being made and provided for us, but from there we could choose to interact with it however we liked. MySQL is an open-

source relational database and the second most used database in the world behind Oracle [1].

Relational databases are composed of several tables which can be linked together. The tables have rows and columns where the rows are the entries into the database and the columns are parameters which make up the entry. Every MySQL table has a unique key called the Primary Key. These keys can be referenced by other tables and then used as foreign keys, and this is what links different tables within the database together [2]. Every database has a schema where the schema is like a blueprint or representation of how the database is structured. The schema shows what each table look like and who owns the tables. Since MySQL is a relational database, this means the schema for the database is rigid. This means that for an entry into the database every single column in a table must be filed in. If the table contains some entries that do not fill in every column, then the memory for these fields will still be used and they will be wasted. Below is an example of a properly set up MySQL database's schema (Figure 4).

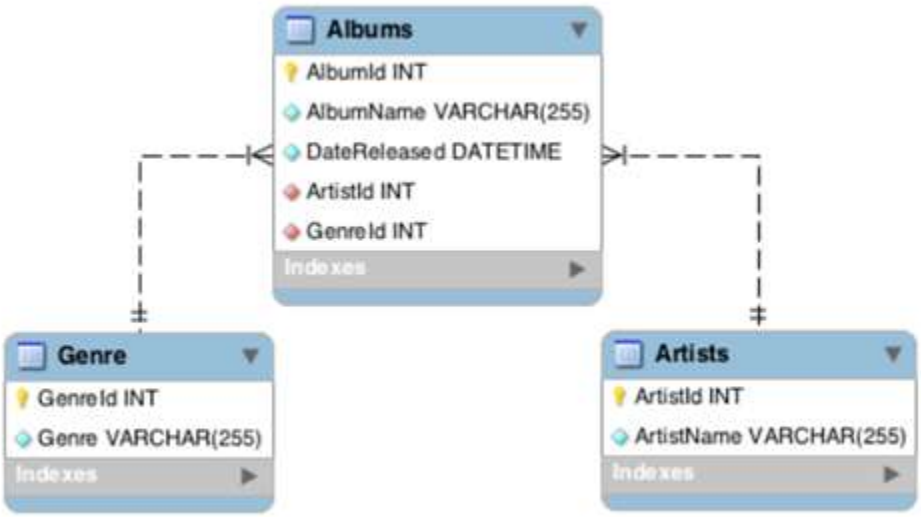


Figure 4: Example of a MySQL Database Schema [2]

MySQL along with being a relational database contains SQL which stands for Standard Query Language. SQL is used to communicate with the database through commands. By using commands such as CREATE, SELECT, and INSERT you can enter data, delete data, or describe your data where SQL will return information from one or more database tables. With a rigid and easy to map out structure it's easy to see why our sponsors want to use MySQL as the database for the TGP website.

The MySQL database is going to be able to be edited in two different locations. One of these locations is the website we are creating, and the other is the community platform Zooniverse. Both our website and Zooniverse are going to allow for entry into the database through CSV files uploads. Data can be uploaded from CSV files through a LOAD DATA INFILE statement. This statement reads the text from the data into the database if the CSV file matches both the number of columns and the data type of each column. Some data though can be edited such as date which are important when it comes to gravestones. Dates that are entered into a CSV file in the month/day/year format can be read and then transformed into the SQL format using the str_to_date() function [3].

The database we are using is being hosted in a server that is run by the UCF Center for Humanities & Digital Research. To connect to the CHDR we all had to download and run a VPN before SSHing to the server. A VPN works by adding a layer of encryption around the data being sent from your device to a server. This makes it so that if someone can get ahold of the data then the data will still be encrypted and therefore secure. It makes it so it is almost as safe as if you connected to the database on LAN, so the VPN is done as an added security measure.

### 5.1.2 Node.js

Node.js is a free open-source JavaScript runtime environment, it is written in C++ and allows us to run JavaScript code on the server. The main purpose of this is to build scalable network applications. It is great with scalable applications because Node.js runs on a single thread and is event driven meaning it is non-blocking, it almost never waits for an API to return information. Node.js is typically used with MongoDB and NoSQL libraries, but it can be used with a MySQL database.

Express is an open-source back-end framework within Node.js with the purpose of creating a fast and reliable API. For our website Express is going to be used as the main backend technology. Express offers us tools to make writing Node.js apps easier. One of the tools is that it minimizes the amount of code we must write regarding routing and makes it maintainable. It is also easier to comprehend and much harder to make mistakes in Express than if you were to write the backend in pure Node.js. Since our front end is being written in React this means that the entirety of our project is being written in JavaScript which is one of the big reasons why we went with Node.js with an Express.js framework.

### 5.1.3 React

For the frontend framework we decided immediately that we wanted to work with a JavaScript framework. JavaScript is a language that we were all moderately comfortable with and it is a popular option for most modern sites. Both Angular and React could have worked well with the web app, but in the end, we decided to not go with Angular. One of the main reasons for this being that Angular has an extremely high learning curve compared to React. Another reason is that Angular is styled in CSS sheets and my group is not a big fan of CSS, while all the styling for React is written within one file. React uses JSX which is an extension of JavaScript that allows you to write HTML code in JavaScript. According to State of JavaScript, React is the most popular front-end framework from 2016 to 2020 having the most people say they have used React and want to use it again [4]. This plus with the team's experience using React made it an easy pick for our stack in the end.

## 5.2 Current TGP Website

We were given a previous website sent to us by our sponsor that another Senior Design team has been working on. Information on this site is not maintained by us and subject to change. The website is hosted on: http://thegravestoneproject.com/. The website uses WordPress, but we will only be linking the project to this site. In this section I will describe the pages we were given and their intended purpose. The information will be presented as text or either a picture of the current page and below that will be its intended purpose.

### 5.2.1 Home Page

*The Gravestone Project (TGP) is a digital humanities collective that brings together scholars, taphophiles, students, writers, teachers, and others interested in history, literature, and the arts, to think about the various ways that people memorialize the dead. Our goal is to initiate projects that allow academics and the public to work together in knowledge creation.*

*TGP has two new initiatives that will guide our project moving forward: TGP Exhibits and Memorial 20/70. Experts and scholars will work to curate public-facing digital* ***TGP Exhibits****, often containing community-generated elements*

*such as crowd-sourced photo galleries. Many exhibits will focus on gravestones and cemeteries, but others will call attention to other forms of memorialization such as commemoration in literature or in oral histories.*

***Memorial 20/70** is a major initiative that will allow us to gather data related to the commemoration of deaths in the centennial decades (20s) and semicentennial decades (70s) between the 1620s and 1920s. While we'll collect images of gravestones and memorials, we'll also collect obituaries, asylum records, plantation records, funeral cards, and other texts that will allow us to consider the relationships between different forms of memorialization. Memorial 20/70, which will initially focus on the United States, the Caribbean, and the United Kingdom, will include a major crowd-sourced transcription project through the Zooniverse platform. In addition to TGP Exhibits and Memorial 20/70, TGP hosts an informal scholarly journal, Grave Notes, and photo galleries of graveyards and notable memorials visited by collaborators.*

Figure 5: Home Page Information of Existing Site [5]

The website contains a lot of historical information for a user to observe when they visit. When a user first logs on to the website, they are met with an about section that basically explains the different initiatives that will guide our project moving forward. This is important for users to first view because it gives context behind the movement of TGP and the passion behind the project itself (Figure 5).

### 5.2.2 Memorial 20/70 Phases Page

***Phase 1A** of the project will focus on the datasets current project collaborators already have in hand: photographs—primarily of English-language gravestones, memorials, and obituaries—as well as textual records of no longer extant memorials. The geographical focus for Phase 1A is the United States, the United Kingdom, and the Caribbean. We will use these datasets to develop a robust system for the crowd transcription and classification of photographs on the Zooniverse platform. During Phase 1A we will also work on the design of a database for the storage and searching of Memorial 20/70 data.*

*During **Phase 1B**, we will beta test our classification and transcription system with volunteers. We will also develop and beta test a submission system so*

*that others can contribute photographs in the future. At this stage we will also test the data output from Zooniverse to ensure that we are generating the most broadly usable data.*
*During **Phase 2A** we will make our Zooniverse page public to increase the volume of transcriptions and classifications of our data.*

*During **Phase 2B** we will open up submissions so that we can expand the scope of our datasets, as well as the type of memorial objects we can consider (including, for example, mourning jewelry and memorial service cards). Initially, our calls will be to expand our existing datasets–that is, of memorial objects and text-bearing objects related to the United States, Britain, and the Caribbean.*

*During **Phase 2C**, we will also seek project partners beyond the United States, United Kingdom, and Caribbean. Our goal is to create a system that will be flexible enough to accommodate a wide range of memorial texts and objects, and also possibly extend into other languages and geographical regions. The directions that we are able to expand will depend largely on the collaborations we are able to build at this stage.*
*During **Phase 3**, we will expand our data set parameters beyond the United States, the United Kingdom, and the Caribbean.*

*We will create a searchable database that will make use of all of the data gathered, classified, and transcribed during Phases 1 and 2. We also will begin to disseminate the findings of Phases 1 and 2.*

Figure 6: Memorial 20/70 Phases Page Information of Existing Site [6]

The main bulk of the project and it's milestones to be met are listed on the "Memorial 20/70 Phases" page. This page lists mainly the description of milestones in the format of "Phases". It's filled to the brim with information about the future of the project and can be useful to users if they want to peer into the future of the TGP. Our expected contribution to the project will be most likely concluded around Phase 2A/2B (Figure 6).

### 5.2.3 Grave Notes Page


Figure 7: Grave Notes Page on Information of Existing TGP Site [7]

Our sponsors are not only instructing us on how to complete our required parts, but themselves are very passionate about the gravestone project. In the “Grave Notes” section there is listed a multitude of links to articles about eighteenth and nineteenth century graveyard information and the cultures buried within them. This section of the site is presumably a way to inform people about this history by providing a select number of articles (Figure 7).

### 5.2.4 Galleries (Cemeteries) Page

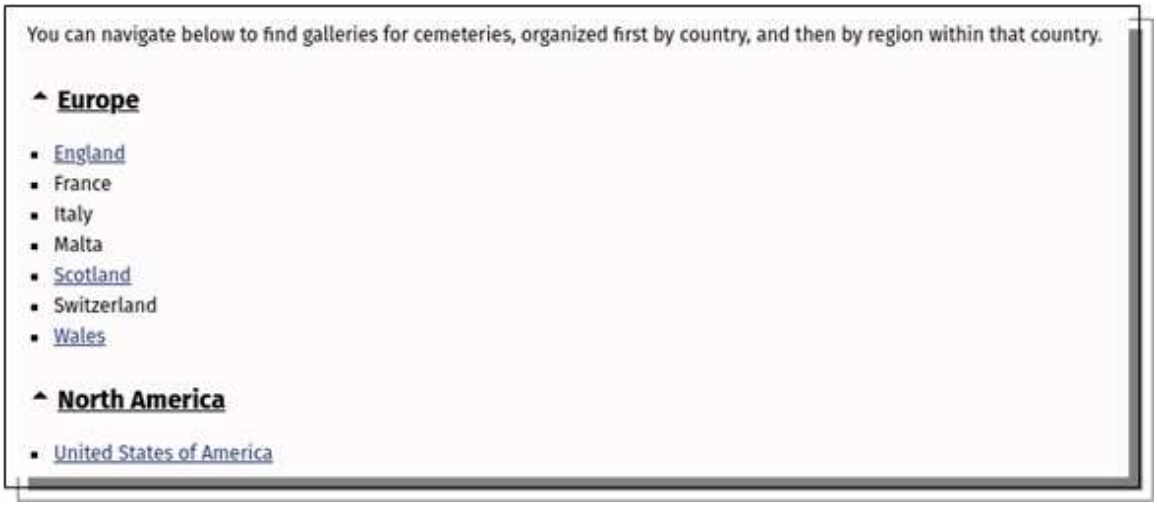


Figure 8: Cemetery Selection Page of Existing TGP Site [8]

The “Galleries” section is presumably where users will be able to browse different cemeteries from all over the world. The “Galleries” section itself is split into two different sections: Cemeteries and Notable Graves. If we head to the Cemeteries section, we are met with being able to choose to view cemeteries, organized first by country, then by region (Figure 8). If we were to select one of these options, for example England, it would take us to a page and present us with the information listed similarly to Figure 9, shown below.


Figure 9: Portion of Cemetery Selection (England) of Existing TGP Site [8]

### 5.2.5 Galleries (Notable Graves) Page


Figure 10: Portion of Notable Graves Selection of Existing TGP Site [9]

If we head to the Notable Graves section, we are met with being able to choose to view cemeteries that are marked notable. By “notable graves" we mean not only graves of poets and novelists and other traditionally “literary” authors, but also of philosophers, painters, scientists, and other prominent authors and thinkers and artists who influenced eighteenth- and nineteenth-century British and American literature–as well as friends, family members, and patrons that figured prominently into the lives and works of literary authors. This is presumably where users will go if they want specific information about an individual's life and the history associated with it (Figure 10).

### 5.2.6 About (Methodology) Page

*The Gravestone Project (TGP) is a collective that seeks to center the methodologies of the humanities and arts to investigate aspects of 17th- to 19th-century memorials and memorial cultures. We particularly focus on the ways that literary scholarship, cultural history, and the arts can enrich our understanding of the objects, spaces, ideas, and beliefs that characterize approaches to death and memorialization.*

*As a collective, TGP houses within it multiple initiatives that consider different aspects of, and different methods of responding to, memorialization. Although each project has distinct parameters, our intention is that the projects will enrich one another by facilitating conversations and the sharing of data.*

*Our newest project is Memorial 20/70, which will collect and analyze memorial objects and texts related to deaths in the centenary and semicentenary decades between the 1620s and the 1920s. We have just published the second volume of Grave Notes, a scholarly journal that features essays investigating the connections between burial sites, culture, and the arts. We also house galleries of photographs of cemeteries that contain pre-1900 memorials, as well as photographs of the gravesites of individuals of significance to history and/or the arts. More information about the methodology of different TGP initiatives can be found on the pages for those initiatives.*

*These and other projects currently in development within the Gravestone Project collective aim to facilitate the exploration of educators, writers, students, artists, scholars, and all others who are interested in cultures of memorialization.*

Figure 11: Methodology Portion of About Section Page [10]

In the about category, there are a multitude of selections that each correspond to a different piece of information about the website. This information ranges from the methodology of the project to the personal reasons of everyone involved. If we were to select methodology, this brings us to a page describing the cultural reasoning behind The Gravestone Project (Figure 11).

### 5.2.7 About (Origins) Page

*Long enamored of old gravestones, Emily B. Stanback visited a small cemetery in Barlaston while conducting research at the Wedgwood Archive in July, 2012. Struck by one particular gravestone, she took photographs of it (above) and transcribed the epitaph:*

*"Afflictions sore, some time I bore*
*Physicians were in vain,*

*Till god did please Death should me seize,*
*To ease me of my Pain."*

*Fascinated by the textual possibilities of this epitaph—and of gravestones in general—Emily began to talk about collaborative possibilities with Polly Rowena Atkin, who she knew was also long interested in cemeteries, and especially how they are depicted as physical spaces in literature.*

*Polly and Emily realized that a searchable database of gravestones and epitaphs would be interesting to the public and useful to academics and educators across disciplines. They also thought that photo galleries of historically- and culturally-significant graveyards would be a broadly interesting—and useful—resource. From the start, Emily and Polly were interested in the potential of crowd-sourcing, and also in finding ways to account for modes of memorialization beyond gravestones and memorial plaques. By 2013, Polly and Emily had begun to make trips to graveyards to document them, and TGP had begun to load photo galleries—sometimes with epitaphs Emily transcribed. Soon after, that they launched Grave Notes, TGP's informal scholarly journal, under the leadership of Julian Knox. For several years little happened aside from graveyard visits, new gallery pages on TGP's website, and most recently, a new edition of Grave Notes.*

*TGP entered a new phase in 2020 through the addition of several new collaborators: Rebecca Bullard, Amy Giroux, Rebecca Schneider, and Stephanie Walters. In 2021 we also were joined by Laura Kremmel. Through these new partnerships we developed two initiatives that will shape TGP in the years to come: Memorial 20/70 and TGP Exhibits. Memorial 20/70 will allow us to work towards the database Polly and Emily had envisioned a decade ago. Both initiatives will allow TGP to more fully integrate crowd-sourced material into our project, and will allow us to work towards our goal of creating material of interest to the broader public, in addition to students and scholars. Importantly, Memorial 20/70 and TGP Exhibits also will allow us to begin to more fully account for modes of memorialization beyond gravestones and memorial plaques—a particularly important goal as historically, the vast majority of burials have not been marked in stone.*

Figure 12: Origins Portion of About Section Page of Existing Site [11]

In the origins section, users are presented with an origin story of The Gravestone Project. It is important for users to be able to read this because it puts into perspective the passionate collaborators of the site. This is what we hope to gain most from this project is give people that are passionate about this subject a place to store their discoveries. A lot of times these gravestones and the story behind them are lost to history and we aim to remedy that in any way we can (Figure 12).

### 5.2.8 About (Project Collaborators) Page

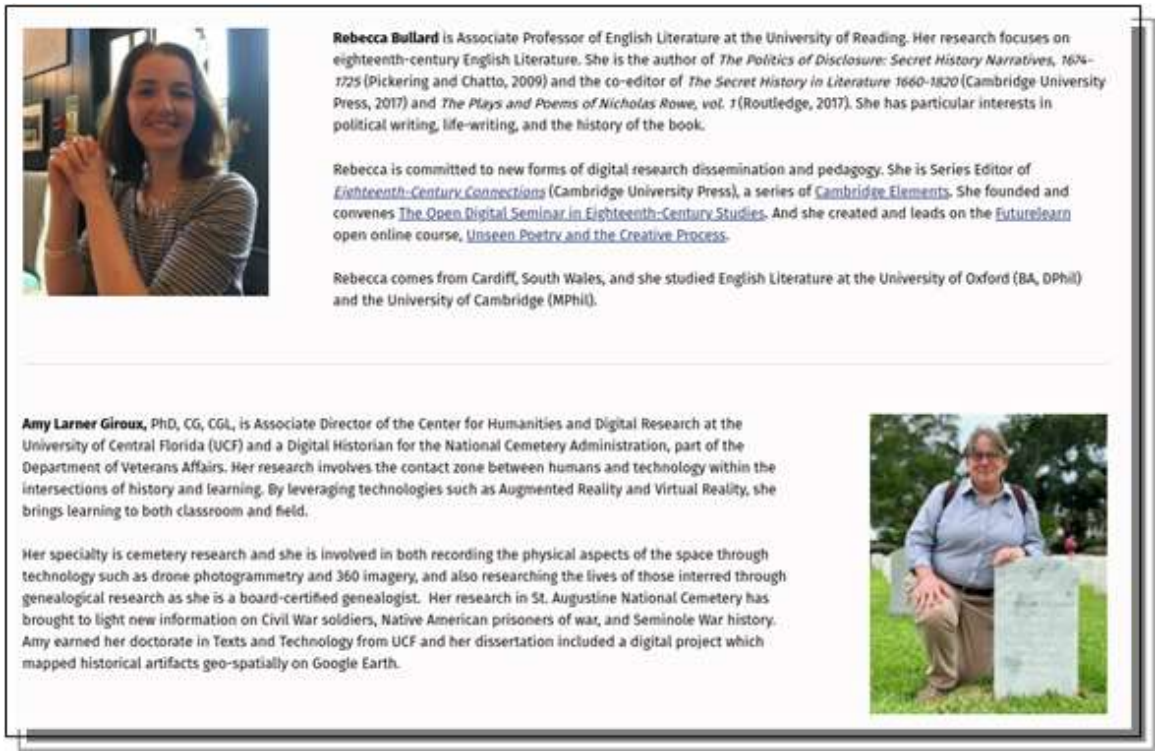


Figure 13: Project Collaborators Section of About Page of Existing TGP Site [12]

The final functional section of the current site is the Project Collaborators page and the Contact Us page, but this page is not functional at the time of writing. This page is there to let the users know who all is contributing towards the site and collecting and processing the information to store. This can be useful if a user wants to contact a collaborator to verify historical information or any of the sort (Figure 13).

It's important to show all these pages and understand their functionalities as this is the first thing users are going to see when they visit the website. Our site will, presumably, be hosted as a tab on the information bar at the top of the website and where users will click to be redirected. Without this current website our website

would quite possibly be out of context and our users would not have previously received all the historical information as they will with the current website.

## 5.3 Related Work

### 5.3.1 Memorial 20/70

Memorial 20/70 is a collaborative project that seeks to investigate shifting practices and patterns of commemoration across a roughly 300-year period from the 1620s to the 1920s. The project examines memorial texts and objects related to deaths during the centennial and semicentennial decades of the 1620s, 1670s, 1720s, 1770s, 1820s, 1870s, and 1920s. These decades will serve as touchstones that can allow us to explore important changes in memorial practices across time and geography. The primary aim of the project is to investigate text-bearing objects that individuals and social groups used to commemorate the dead. This research will also illuminate other areas of inquiry, including illness, religion, childhood, old age, memory, migration and colonization, ideas of identity and heritage, and conceptions of kinship and community.

We intend to include a broad range of memorial objects in our project to enable the study of the relationship between different commemorative forms and practices. We wish in particular to expand beyond gravestones and memorial markers because the use of such forms of memorialization has been highly dependent on social and economic capital. Most gravesites from the 1620s to 1920s remain unmarked, but some of these have been memorialized in records (e.g. asylum or parish records), in letters or journals, or in oral histories that were later recorded. Our project investigates these records as well as text-bearing objects such as funeral cards, mourning jewelry, obituaries, and parish records, in addition to church monuments and gravestones. We aim to be able to compare the ways in which individuals were commemorated in different media, across a range of geographical areas, at distinct points in time during our period of inquiry.

From a digital humanities (DH) perspective, one of our most ambitious goals is to create a system for linguistically, thematically, geographically, and temporally analyzing side-by-side forms of data that are not typically considered alongside one another: photographs of gravestones, obituaries, mourning jewelry, and asylum records, for example. Ultimately, we hope to provide data that will allow for

analysis and use by members of the public, as well as scholars and educators across several fields–history, literature, art history, sociology, linguistics.

### 5.3.2 CemetARy

A current senior design project by individuals in the same class. This is also a project by the Gravestone Project. The goal is to create an augmented reality mobile application that is cross platform that allows switching to 3D augmented reality navigation once a user approaches a headstone at a cemetery. It uses OCR - Optical Character Recognition - scanning to read headstone text for searching. It will also have a preliminary administrative website for database manipulation. They are also hoping to reach the stretch goal of creating a tour mode. The project will be completed by December 2021.

### 5.3.3 The Gravestone Project Exhibits

The Gravestone Project Exhibits are informed in their conception, design, and implementation by experts and scholars. Most will feature a significant community engagement element, and all are intended for public audiences. Our goal is to create exhibits on topics that will interest a wide range of audiences, but can also be used in middle-school, high-school, and college classrooms. The first TGP exhibits will launch in August/September 2021.

## 5.4 Metadata Standards

### 5.4.1 What is Metadata?

Metadata is a core ingredient used in many enterprise-wide applications. It is described as being a set of data that gives information about other data. Metadata is the backbone behind almost every application that needs to store data and the information with it, in an easy, describable way. It is used in things like organizing electronic resources, providing storage of identification methods, and archiving data along with its specific information. It is especially important in allowing users to query and retrieve this data for observation purposes or for the telecommunication activities where owners of a product can see traffic of their product and determine relevant choices based on that. It does this by summarizing the data into basic information which makes gaining traffic information and working

with specific data easier. Metadata also comes with the benefit of being highly agnostic and reusable as there most likely exists a metadata set for anything someone could possibly need.

### 5.4.2 Uses of Metadata

Like described previously, metadata has an abundance of uses ranging from statistics to transcribing data into basic information for users. Some basic examples of metadata are stored in a database: author, date created, date modified, file sizes, location information, place of death, etc. [13]. Metadata is also a great way to store similar data together for use in searching, such as a search engine, or data storage. Browsers use this as a way to display relevant information about a web page and to find any other pages similar to it by using keywords. The keywords can be used to match the current site to other sites with similar keywords and vice-versa. While designing a web page, programmers use the <meta> tag of HTML to describe the metadata within an HTML document that browsers fetch to display this information [13]. Figure 14 below is an example of this.

```
<meta charset="UTF-8">
<meta name="description" content="Place to store animations">
<meta name="keywords" content="Flash, Animations, Cartoons">
<meta name="author" content="John Doe">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Figure 14: Example of HTML Meta Tags

This metadata is used by browsers, search engines, and other web services [14]. The description, author, and viewport can be primarily viewed as how the search engine will display the website to users. The keywords are, again, how a search engine determines how relevant a specific search is and how useful it will be to display it.

Some more specific examples of uses for metadata include [13]:

- Methods of data generation
- Purpose of the data
- Time and date of creation
- Creator or author of the data

- Location on a computer network where the data was formed
- Standards utilized
- File size
- Data quality
- Source of the data
- Process used to create the data

### 5.4.3 Types of Metadata

Metadata can be mapped many different ways and describes many different aspects of data. Although there are many, metadata has been boiled down to three specific types: descriptive, administrative, or structural.

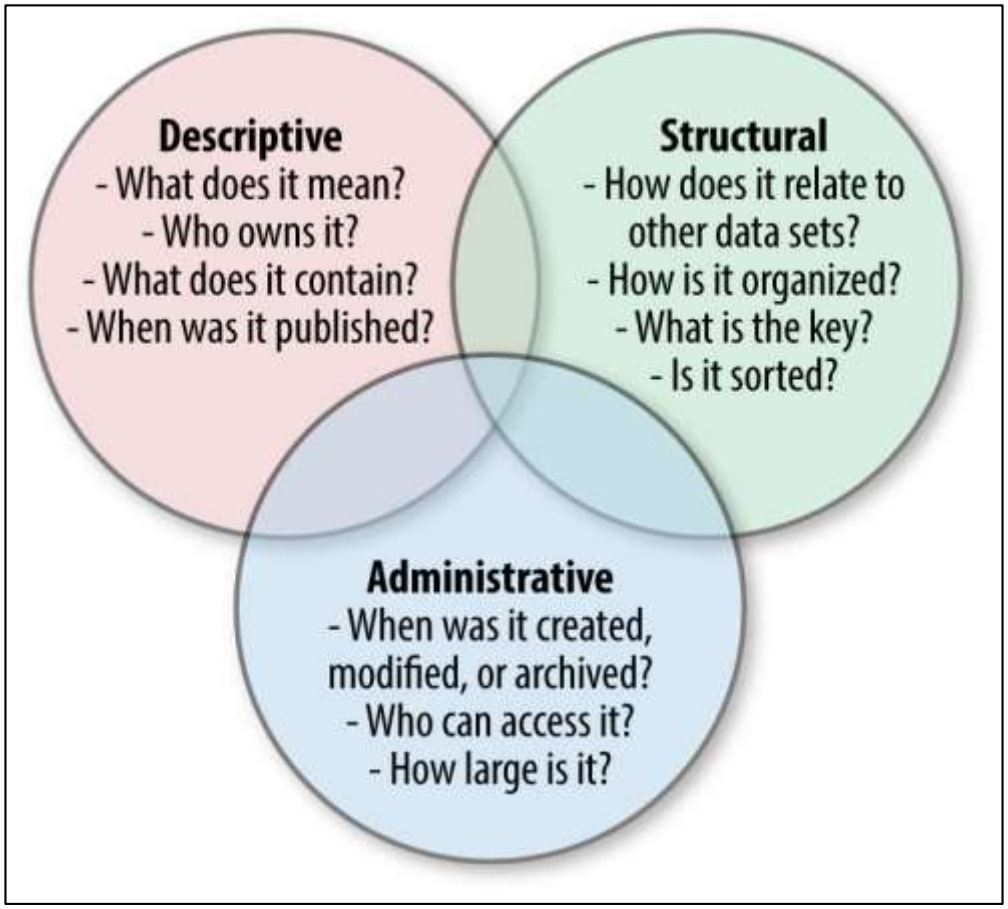


Figure 15: Venn-Diagram of Different Metadata Types [15]

From the image above (Figure 15) we can see a brief overview of the different types of metadata. To give a few examples of each and give a more in-depth description would better visualize how each type differs and their use case.

Descriptive metadata is metadata used for describing an object. Things like an author and date created are used to describe a book, or taste of a fruit and it's color to describe this fruit. This is especially useful for our project because it allows us to store an artifact using this type of metadata. An artifact can have the attributes including place of origin, discovery date, etc. This is especially useful for items we want to be searchable, as users can search for a specific keyword/attribute instead of the item's full name. Below are a few examples of descriptive metadata [15].

**Descriptive Metadata Examples:**

- Unique identifiers (such as a book ISBN)
- Physical attributes (such as a building dimensions)
- Author or creator of a book, as well as its title and date
- Web Page attributes (such as author or creator, keywords, author)
- Artifact Information (owner's place of death, place of birth, etc.)

Administrative metadata is metadata used for describing the digital footprint of an item. This digital footprint can include attributes such as: time created, usage rights and intellectual property, owner of said item, those allowed access to said item, etc. This is useful for storage as we can see the many importance's of this information [15].

**Administrative Metadata Examples:**

- A Creative Commons License
- Photo creation date
- Entity creation date
- Administrators that can access data
- Information necessary for decoding and rendering files

Structural metadata is metadata used for describing how an item is organized. Say you have a digital book; how would you know what the contents of each page were

and how they were sorted? This is how structural metadata comes into play to describe to the user an easy way to find this information. Structural metadata can also be used to describe the relationship between items and how they are structured [15].

**Structural Metadata Examples:**

- Page Numbers
- Page Sections
- Book Chapters
- Indices
- Table of Contents
- Note Pages

### 5.4.4 Different Schemas of Metadata Standards

Metadata standards are the different kinds of metadata and their applications. There are over 400 “official community” metadata standards widely used, but presumably individuals will create their own for their specific needs. In this section, we will present a few different metadata standards and their uses. The selected metadata standards chosen for this section are not the one we were tasked with using in The Gravestone Project, as that will be discussed in the next section (5.4.5).

Upon researching, almost every profession uses metadata standards in some way. From biology and needing to categorize the organisms of a microbiome, to a movie renting service needing to organize and sort their movies by attributes, metadata does it all. In the example shown below (Figure 16) we show the stages of microbiome analysis and each metadata they use along the way [16].

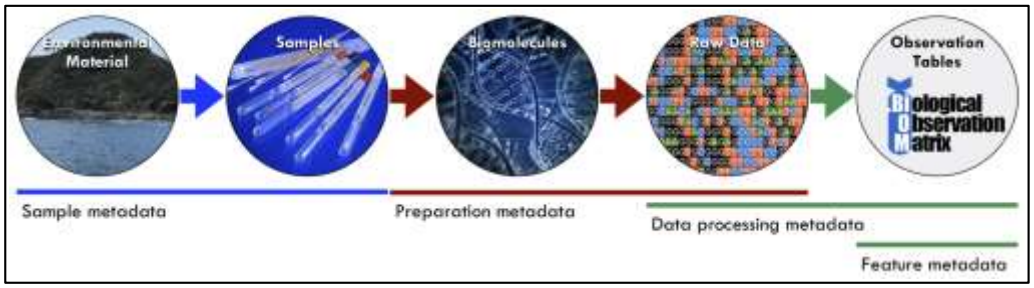


Figure 16: Different Types of Metadata in a Microbiome Analysis [16]

As you can see in Figure 16, there are many different stages to this analysis and each stage uses a different metadata standard! This shows how you can change around data even if it is a part of a metadata already to fit another one; the re-usability of them is a key benefit.

For example, the first stage of this analysis is “Sample metadata”. The site lists this metadata as storing the following attributes of the biological sample:

- Where it was collected (ex. Latitude, longitude, elevation/depth, site name, country, etc.)
- What kind of sample it was (ex. Soil, seawater, feces)
- Properties of the environment during collection (ex. Temperature)
- What kind of state the sample was in

The analysis then proceeds to use more metadata sets that include: preparation metadata, data processing metadata and feature metadata [16].

In this next example, shown below, we can show an example of how metadata is used to store the information about an image (Figure 17). Metadata is great for storing images because, with said information stored, it makes for easy lookup; by name, date, people included in the picture, photographer, etc.

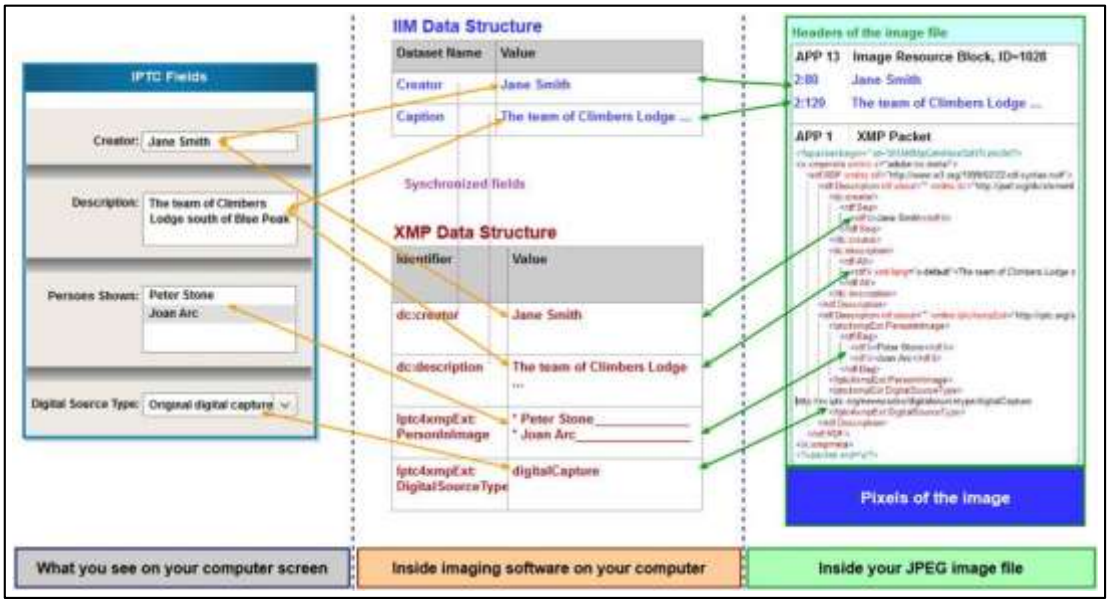


Figure 17: Example Diagram of Storing an Image using Metadata [16]

As you can see above (Figure 17) this is an example of how an image could possibly be stored using metadata. As this is a more programmatic example, you can see the attributes are stored in a key-value pair. This allows for the data to be used in almost any conceivable way and does this read-ably.

The specificity of the data shown in this specific example is:

- Creator of the image
- Caption/Description of the image
- Person(s) included in the image
- Source type of image

### 5.4.5 Dublin Core

Dublin Core is another metadata standard and is the one we were specifically tasked with using. It is one of the most widely used metadata standards as it is simple and highly reusable as it contains generic data that can be fit to almost any project. In our case it is perfect to be able to store artifact/gravestone information as there exists in it all the slots available for information we would need to store. Figure 18 is an example of the Dublin Core metadata standard:

| Identifier | Definition |
|---|---|
| Title | A name given to the resource. |
| Creator | An entity primarily responsible for making the content of the resource. |
| Subject | The topic of the content of the resource. |
| Description | An account of the content of the resource. |
| Publisher | An entity responsible for making the resource available. |
| Contributor | An entity responsible for making contributions to the content of the resource. |
| Date | A date associated with an event in the life cycle of the resource. |
| Type | The nature or genre of the content of the resource. |
| Format | The physical or digital manifestation of the resource. |
| Identifier | An unambiguous reference to the resource within a given context. |
| Source | A reference to a resource from which the present resource is derived. |
| Language | A language of the intellectual content of the resource. |
| Relation | A reference to a related resource. |
| Coverage | The extent or scope of the content of the resource. |
| Rights | Information about rights held in and over the resource. |

Figure 18: A Table of Dublin Core Metadata Standards

Dublin Core is a set of 15 metadata elements aimed at facilitating discovery and search-ability for search engines. As you can see above (Figure 18), we can list these elements by an identifier and a description to describe what each identifier will include. Dublin Core happens to be the most common metadata standard. Dublin Core started as a way for authors to increase the discover-ability of their web product, it has since grown and matured. Dublin Core was given the opportunity to rise to the top because of the huge community support behind it. It is intended for non-catalogers that only require a simple schema to follow to store their data, as each element is easily understandable and requires no technical knowledge.

As we are using Dublin Core for our project it is important to understand what kind of information we plan to input using this schema. For us primarily we are going to be using primarily artifact information, as well as grave sites. Dublin Core allows us to seamlessly reuse it instead of having to create our own schema. An example artifact sampling using Dublin Core could be as follows:

- Title: Pigeon Stone
- Creator: N/A
- Subject: Native American History
- Description: Artifacts of Native American life of the 16th and 17th centuries on the Eastern Coast
- Publisher: N/A
- Contributor: N/A
- Date: 16$^{th}$ and 17$^{th}$ century
- Type: Artifact
- Format: N/A
- Identifier: N/A
- Source: Native American Research Center
- Language: Cherokee
- Relation: Cherokee Tribe
- Coverage: Eastern Coast
- Rights: Cherokee Preservation Foundation

As Dublin Core is so flexible, can have empty fields and still have a conclusive piece of data. This is useful, especially to us, because we will be receiving submissions from users all around the world. If a user does not know a specific piece of information, this shouldn't invalidate his submission, rather our system will be able to store and handle it. This is great because, statistically, it is likely users will not know every piece of information about their submission.

## 5.5 Google Analytics

Google Analytics is a free service that can track data such as how users found your website, the website's bounce rate, and pages per session. Site search is an important feature of Google Analytics that we want to use to analyze our website. Site search is used to help understand how the search feature is being used and what type of artifact information that users are searching for. Google analytics will help us improve our website by tracking how long users spend on each site, see which pages they navigate to, and which pages are rarely visited. Google analytics is also a powerful tool for security. It will allow us to view website traffic and the location the traffic is coming from. This can obviously be spoofed with a VPN or virtual private network. However, if we see a ton of traffic all at once or originating from an unusual location, then we can look at it more closely to monitor if a potential hacker is targeting our site to steal information or for a DDOS attack. More information about this will be in the threat model section. We will also be able to see what devices and operating systems each site visitor is using. This is helpful for knowing what browser, device size, and operating system to prioritize and test is working as expected. Many websites do not provide support for all web browsers, especially older versions. Instead they will track the most commonly used like Google Chrome and have developers test functionality on it.

For this to work, we will need to import the Google Analytics tag to the top of our code. For React we have a google analytics library we can import called ReactGA (Figure 19) [17].

```
import ReactGA from 'react-ga';
ReactGA.initialize('UA-000000-01');
ReactGA.pageview(window.location.pathname + window.location.search);
```

Figure 19: Usage Code for Initializing GA and Tracking Pageviews

Then through the analytcs.js we will send a virtual page path for whenever the search is performed on our website. This is done through customizing the google analytics tracking code. An example of this would have the URL being www.example.com/search and the analytics tag would be: analytics.js: ga('send', 'pageview', '/search/?=WWI+Female.

## 5.6 Google Forms

Google forms is the tool that TGP currently uses to collect artifact information. This is presumably because of the ease of access of Google Forms. This will mean most people will be able to submit artifacts without any advanced technical knowledge. The sponsor has requested that we continue to use the Google Form as the main form of user import for the website. With this being the case, it means that regular users will have no direct access to edit the database and instead the admins will go through the forms and add the data on a screen they only have access to. Along with this the form is going to be embedded directly into the website using an iframe tag.

To expand on this we wanted to give a few examples of how the Google Form will look and it's input fields within the site. The specificities of our input fields are subject to change as we may want to include or remove specific metadata.


Figure 20: TGP Submission Form Header (Mock-up)

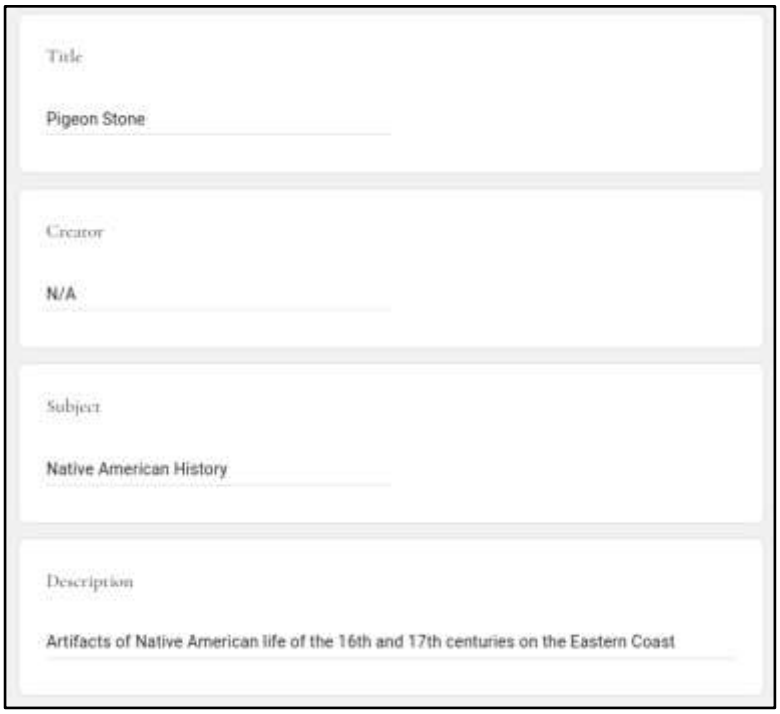


Figure 21: TGP Artifact Submission Fields Part 1 (Mock-up)


Figure 22: TGP Artifact Submission Fields Part 2 (Mock-up)

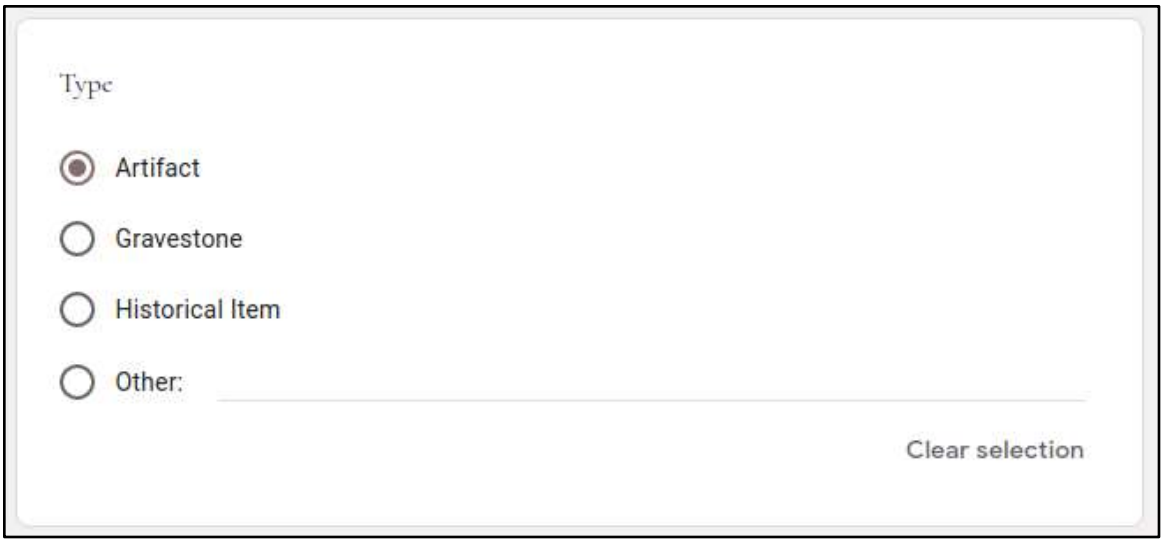


Figure 23: TGP Submission Type Information (Mock-up)

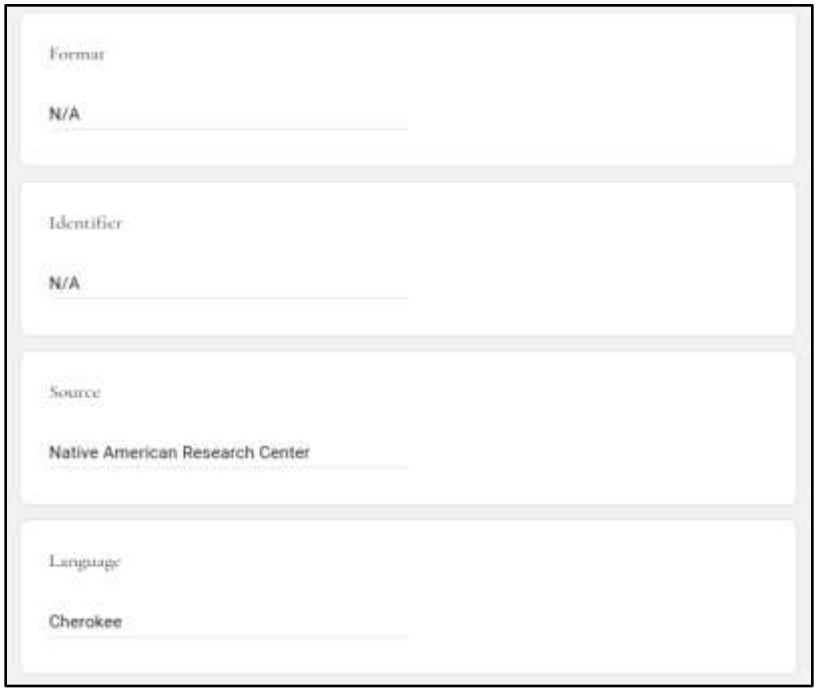


Figure 24: TGP Artifact Submission Fields Part 3 (Mock-up)

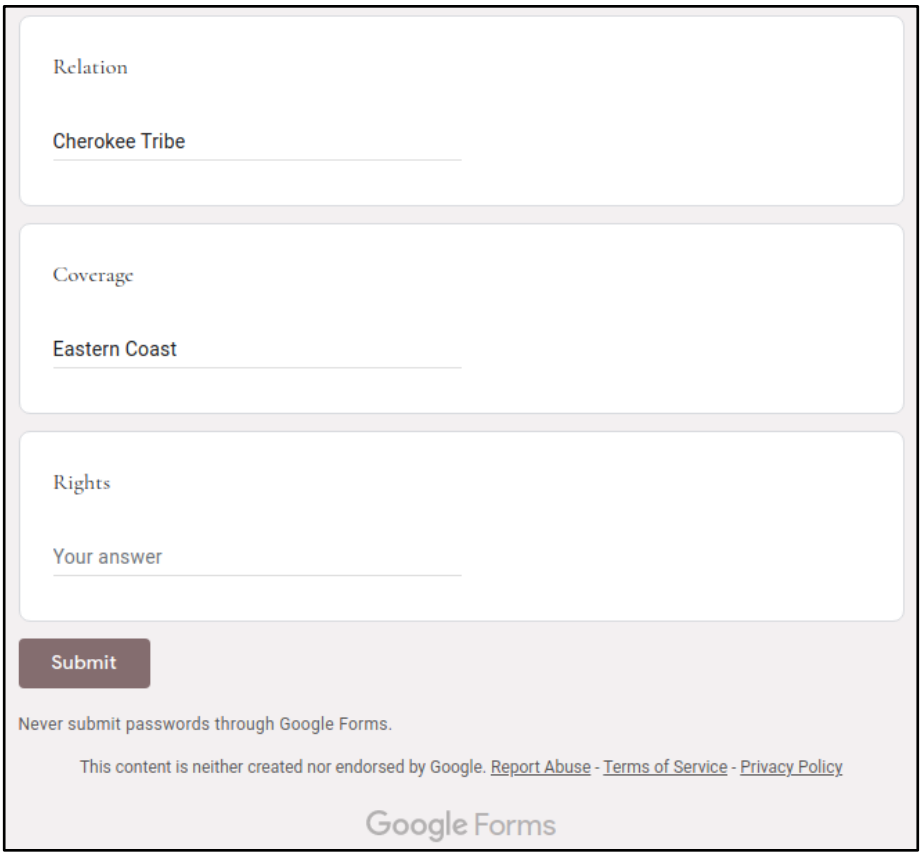

Figure 25: TGP Artifact Submission Fields Part 4 (Mock-up)

The Google Form's purpose is to upload a single submission at a time. While this is most likely optimal in most cases for our project, our sponsor suggested we look at methods of batch uploading. We believe using Excel for this will serve us well. It will function similarly to the Google Form, but rather submissions will be stacked in rows and submitted.

## 5.7 Indexing

Indexing is a technique used to minimize the amount of time it takes to locate data during a database query. Indexing makes it so the database does not have to potentially through every piece of data for a query. The database we are using for this project utilizes a $B^+$-Tree which is also the most popular form of indexing for databases. A $B^+$-Tree is a self-balancing tree that is made up of keys that increase from left to right. Each key has up to two children's nodes which eventually branch out to the data (Figure 26).

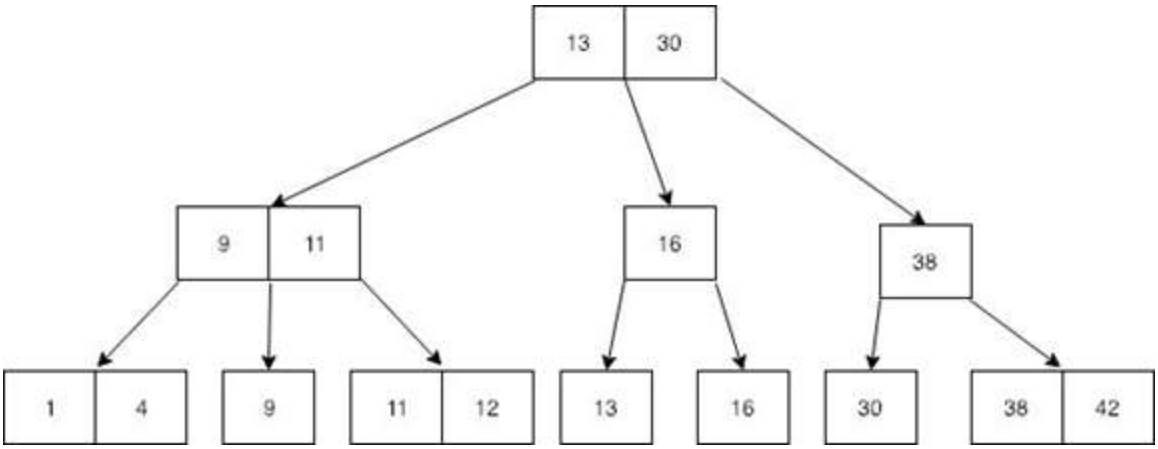

Figure 26: An example of a $B^+$-Tree

A $B^+$ is used for these databases over other indexing methods because of the fact you can store more than one value in a node. This decreases the amount of time by decreasing the total distance traveled from the root node to the searched for value [18]. This makes it the best indexing method for databases even when some of them have the same runtime when it comes to insert, search, and deleting.

## 5.8 UI Libraries

A UI component library is a ready to use set of components that help determine the look and feel of a website. A library helps to determine the style of buttons, forms, user inputs, and tables. The library also helps with coding in general as otherwise the user would have to create the look for all these components themselves, it speeds up the development process. There is also more of a guarantee that the components will look good as these libraries are popular and used repeatedly in thousands of websites.

A UI component is still customizable through some details such as color, but it establishes the main style. Since this project is using React for the front-end, we must use a React UI component library. Ten of the most popular React UI component libraries are in the table below [19].

| Name | Stars |
|---|---|
| Material UI | 54,000 |
| Bootstrap | 17,000 |

| | |
|---|---|
| Rebass | 6,000 |
| Semantic React UI | 10,600 |
| Fluent UI | 8,500 |
| Ant Design | 72,000 |
| Blue Print | 15,000 |
| Evergreen | 9,000 |
| Chakra | 4,000 |
| Shards | 5,000 |

The stars here are as an indication of the number of people who rated the UI component library as being useful, they would recommend it to others. We wen through the list above looking at how each library looked because the website we are building is going to connect to the current TGP WordPress website. Our website is to match the same theme and layout as the current website, so we need the components to as well.

We also looked at how active each of the community is within GitHub through pull requests and how many issues get closed. This is important because if the community is active, it means that there are less likely to be bugs, fixes to any bugs will come out quicker and there is also more likely to be more new features added. This is also mixed with the fact that React sometimes releases multiple versions within one month. After looking through the different options we have decided to go with Bootstrap. Part of this is because we are familiar with it and know that it will match well with the current TGP website.

We decided that we did not want to try and learn a different UI component library unless we looked at one that we all agreed looked much nicer and would blow bootstrap out of the water.

## 5.9 JSON

JSON which stands for JavaScript Object Notation and is used to transport the data in a web application from the front end to the back end. JSON stores data as a string which means for us to use JSON on our front or back end we must convert

the string into an object. There are two different functions within JavaScript that handle this. JSON.parse() which is used to change a JSON string into an object and JSON.stringify() which takes a JavaScript object and transforms it into a JSON text string [20]. Figure 27 is an example showing how the two functions work [21].

```
const myObj = {
  name: 'Skip',
  age: 2,
  favoriteFood: 'Steak'
};

const myObjStr = JSON.stringify(myObj);

console.log(myObjStr);
// "{"name":"Sammy","age":6,"favoriteFood":"Tofu"}"

console.log(JSON.parse(myObjStr));
// Object {name:"Sammy",age:6,favoriteFood:"Tofu"}
```

Figure 27: Example JSON Data Transfer Code

This can also be used outside of a JavaScript environment because many other languages such as PHP have the ability to parse into JSON. Because of this JSON has become very popular and is used in most websites. Another reason it is so popular is that JSON is much faster than XML. XML is harder to parse and must be parsed by an XML parser. XML also doesn't have the ability to use arrays and uses end tags making it much larger than JSON. This shows how JSON is just an easy choice for our applications and why it quickly overtook XML as the language to use for data transfer. Along with this the front and back end are written entirely in JavaScript this means we can communicate between the front and back end without having to translate the code. Different databases including MySQL even support the JSON data type.

## 5.10 Data Mapping

For this project we need to integrate a map that has markers popup based on the results of an SQL query. One option is Google Maps. Google-map-react has 5.3k stars on GitHub and has a size of 111kb [22]. Google Maps has advanced features

such as geolocation and traffic data, but for this website all we need are pins/connecting pins on a map, so the features are not useful for us. Google Maps is also well documented, but it is not entirely free as there will be a seven-dollar fee for every thousand searches that are made.

- **OpenLayers** is a free open-source JavaScript library with the purpose of displaying data on a map in web applications. OpenLayers has 175 stars on GitHub and a file size of 7.91 [22]. One major concern with this is that the React library for Open Layers, so it has not been updated in over two years. Also, there is some concern about the speed this will run compared to other options when looking at the file size.
- **Tomtom** has the same features as the other choices on this loop, but it seems like there are more hoops to jump through to end up with the same product as other choices. Tomtom is also not free, although it is cheaper than Google Maps, but the price is still .50 cents per one thousand requests. Tomtom is not open source, but it does have a good amount of documentation.
- **MapBox** is not open source, but they used to be, and they still release code often, so MaxBox is very well documented. Mapbox has 7.7k stars on GitHub and a size of 35.2 kb. A downside to MapBox is that Mapbox is also not free, it is only free for the first 50,000 uploads and after that every thousand maps that are uploaded will cost five dollars [22].
- **Leaflet** is a free, open-source JavaScript library for mobile-friendly interactive maps with 31.1k stars on GitHub. Leaflet includes features such as zoom, attribution links, and different colored markers than can be up on top of the map. Leaflet itself is also only 38kb of JavaScript making it extremely fast [23]. This can be integrated into the code of the website through two packages, named “leaflet” and “react-leaflet”. React-leaflet is a collection of React tools specialized for Leaflet maps.

In the end our choice came down to wanting to use MapBox, Leaflet or Google Maps. While Leaflet may not have the most features, it is fast, easy to use, it does not include a lot of features we do not need to use, and most of all it is free. On top

of this our sponsor has some familiarity with Leaflet so it is a good choice because it makes it easier on them if they need to update the code in the future.

## 5.11 React-Leaflet

The way React-leaflet works is React renders a Div when it renders the MapContainer and then Leaflet itself renders its layers to DOM. DOM stands for Document Object Model which is an api for HTML documents. This MapContainer is used for every Leaflet map, and it then makes use of the React Context API to quickly make elements of the map available to the children who need it without sending the data down a tree [24]. The MapContainer is what is responsible for rendering child elements, not React. An example of a layer in leaflet is a marker and leaflet describe a layer as anything that moves around when the map moves [25].

```
const position = [51.505, -0.09]

render(
  <MapContainer center={position} zoom={13} scrollWheelZoom={false}>
    <TileLayer
      attribution='&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
      url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
    />
    <Marker position={position}>
      <Popup>
        A pretty CSS3 popup. <br /> Easily customizable.
      </Popup>
    </Marker>
  </MapContainer>,
)
```

Figure 28: Sample Leaflet Marker Code [25]

Leaflet is going to be used in our website with a search feature where the user can select a specific cemetery, or the user can search for specific details such as World War I veterans. Leaflet will take in the results from an SQL query and display the results as markers over a map all still integrated into the website (Figure 28). Along with this information about the grave such as name, dates, military position, and an image of the grave will be presented on the map when a grave is selected. Below are two examples of how the mapping could work, one is with pins over all the selected graves (Figure 29). The other contains a highlight over every grave in a cemetery with pins popping up or the color of the highlight changing according to the search parameters (Figure 30).

Something we do need to worry about is loading too many individual pins onto the map. Leaflet has a tool named maker cluster that helps with this in the library react-leaflet-markercluster. A marker cluster either clusters together or disperses markers depending on how many pins would be in the area. When a cluster is hovered over you can see the outline of the cluster. When you click on it you are then zoomed into the area and then given then can see and click on the pins that were within the cluster. There is still a limit though to the number of pins that can be loaded onto a map even if they are in a cluster. This limit is around 50,000 for Google Chrome, Internet Explorer may have some trouble with the max of 50,000 [26].


Figure 29: Example of Mapping with Pins over Selected Graves [26]

Figure 30: Example of Pins over All Graves in a Cemetery [26]

## 5.12 Zooniverse

Zooniverse is a website with the purpose of people-powered research. Zooniverse is not something that will be embedded directly into the website, but rather a tool that will be used to gather more data to be put into the database. Their goal is to make research that is not typically practical possible which is an extremely helpful tool for when you are working with gravestones. The purpose of Zooniverse and this project is to allow another means of people submitting information related to our artifacts Zooniverse will help us out to get information from users who just want to help, there is no login for this project anyone in the world would be able to use this website to help.

The Gravestone Project is going to use Zooniverse to help with the transcribing and classification of data. Once images are transcribed and classified by the public Zooniverse has a feature that allows for the information to be exported. This information is exported through a CSV which contains a field for metadata. The CSV is going to be exported to the MySQL database where the metadata will be entered directly or directly update the database. This is going to streamline the process of having the community help with the project. Zooniverse is also immensely popular, claiming to be the “world’s largest and most popular platform for people-powered research” [27].

We did not end up setting up a Zooniverse account, but it still can be used as a tool by collecting information and putting it into a CSV file. From there the CSV file can be uploaded into the database by an admin on the website.

## 5.13 Voyant-Tools

Voyant is a free analysis environment that’s purpose is to read web text. The project is intended to help scholars interpret what they are reading in a new light. The main purpose of Voyant for our website is going to be to add functionality to our artifacts. It will be added to look at the text through the eyes of analytic tools as seen below [28].


Figure 31: Example Cirrus Word Cloud

Voyant can be embedded in a collection of independent modules into remote site. This is done through an iframe tag and create an area within the page that Voyant does its visualizations. The module above is the Cirrus tool which takes the frequency of words within a document and puts them into a word cloud where the larger words are the words that are most common (Figure 31) [28]. There is also an option to enter in words you do not want searched for, so if you were using this tool for analyzing a transcript you could tell the module to not look for the word “the”. Other tools that are more helpful and less fun that a word cloud include the “Document Terms” tool which pulls up the most common word in different documents so you could find a feature the two documents on two different people share or “Corpus Terms” which pulls up the most common words in a collection of documents (Figure 32).

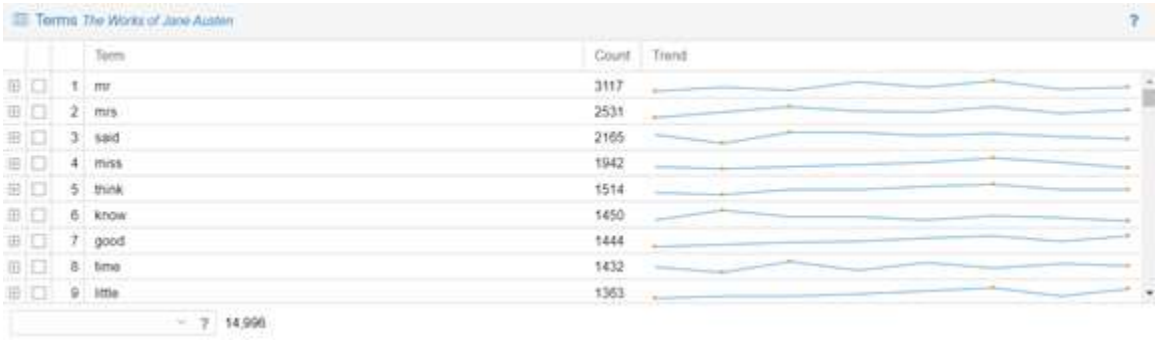


Figure 32: Example of a Collection of Most Common Words in a Document [28]

This is a great feature that we are using, but there are limitations through JavaScript in how much Voyant can do with our website.

## 5.14 Routing Library

A routing library is what allows for the movement between virtual pages in a web application. It will keep the user interface in sync with the current URL, or in other

words a routing library can be used to change a page or use a link without refreshing an entire page. React has many different communities made routing libraries and below is some of the information we found on them.

- **React Router**: This is by far the most popular routing library. This is the best documented library, and it has 43.6k stars on Github and it has a file size of 9.4kb which includes the most features out of any library including hooks.
- **Wouter**: This is a small routing library; it has zero dependencies and is only 1kb in size. This is because Wouter only targets the newest version of React and contains the most used functionality of React Router making it useful for small applications. Wouter has 3.2k stars on GitHub [29]
- **Reach Router**: This was a routing library for React, but it has not been updated in several years. In fact, Reach Routing joined the React Routing project in 2019 to make the routing library with hooks that is in use now [30].
- **Single-SPA**: This routing library has 9.4k stars and a file size of 6.4kb. One of the biggest upsides with this library is made to work with micro frontend, but for this project we are not using them, so this library is not chosen immediately.
- **Found**: Found is customizable and has elements that can be replaced entirely through extensions. Found even uses Redux so it can integrate with your store. As interesting as the library, it has a small community, it is an active community, but Found only has 728 stars and it has a large file size of 17.1kb.

Although React Router is the most used library with the most features, Wouter is smaller in size, but includes the most important React Router components like hooks. With this in mind and the fact that we are making a smaller web application we decided to use the Wouter library for the TGP website. Once Wouter is installed the components of Link and Route must be imported and from there you can use the two types of API Wouter has which are Hooks and Component. Hooks contain the three hooks useRoute, useLocation, and useRouter. UseRoute determines if the current path matches the path that is given, useLocation to return a location object that has info on the current URL, and useRouter that houses the location

hook and matcher function. This last function returns a router object. For the Component API there are four main components which are Route, Link, Switch and Redirect. The route component renders a component when the current path matches the route path. The Route component is not a required component for Wouter unlike React Router and the body can be declared in two ways (Figure 33) [31].
In the end we ended up changing our mind and going back to using React Router's Hash Router. We did this because it has the most features so it is best for use in the future if there are to be features added to the website. Along with that our front end team was most familiar with this router and knew the best website we could make would have to use the Hash Router.

```
import { Route } from "wouter";

// simple form
<Route path="/home"><Home /></Route>

// render-prop style
<Route path="/users/:id">
  {params => <UserPage id={params.id} />}
</Route>

// the `params` prop will be passed down to <Orders />
<Route path="/orders/:status" component={Orders} />
```

Figure 33: Demo Code for Setting Up and Implementing Wouter

The Link will render <a> or an anchor. This means we can go to wherever href is pointing to and of course you can always customize the anchor to say whatever you like. A Switch component will render one route exclusively. Even if there are two routes that could match the Switch will exclusively take the first route that matches. Redirect will redirect the page to a different path, normally this will be based on a condition like a form not being filled correctly or a user trying to access something they do not have access to. An important thing to remember is that the Redirect component uses the useLocation hook component in a lot of cases.

## 5.15 The 8pt Grid System

Before prototyping the website or developing the front end, we put research into some best practices for UI design. One such rule that appeared was the 8pt grid,

a design principle that uses multiples of 8 to define dimensions, padding, and margin of both block and inline elements [32].

The reason the 8pt grid system has become a standard for UI design, to the point that both Apple and Android advise the use of it. This is largely because multiples of 8 scale perfectly to all different screen displays, and because 8 is considered a good unit to work with [33].

There are two methods of the 8-point grid system: hard grids and soft grids. The hard grid version (pictured below on the left) involves sectioning off elements, and placing them relative to a frame element that exists higher up in the design hierarchy. This means components are placed relative to a grid, rather than to each other. The soft grid system (pictured below on the right) involves positioning elements relative to each other, instead of to an overarching grid system.

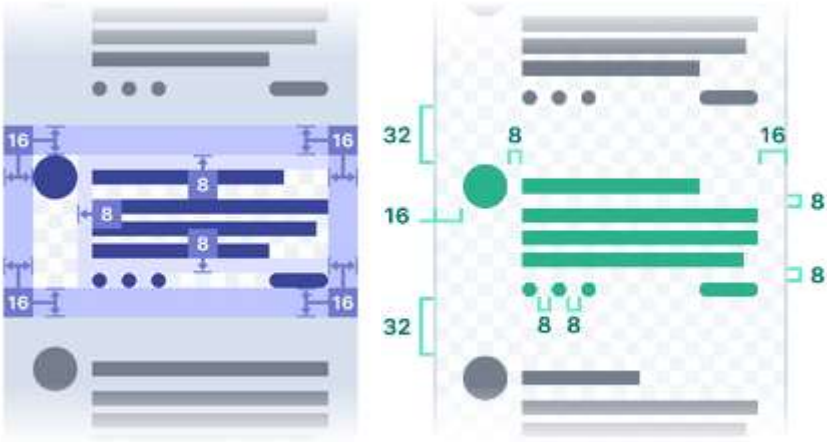


Figure 34: Example of a Hard Grid (left) and a Soft Grid (right) [32]

In consideration of the pages and components that will be designed for the website, both versions will be utilized. Singular components will be designed under the soft grid version, and wireframes of web pages will be designed under the hard grid version.

## 5.16 MySQL vs NoSQL

As the site is primarily a repository full of various artifacts, the main item that the database stores will be the artifacts that have been submitted by users of the site.

Aside from that, the database will also store the users of the site as well as the paths that hold the images that users have submitted. Some metadata that the database also stores include people relevant to the artifact, the location to where the artifact is located or relevant photo has been taken in, tags for a search system, and categorical data for artifacts and people.

As for the software, there are many different types of databases that would be useful for this project. As we are utilizing a “MERN” stack, the most obvious database to use would be MongoDB. MongoDB is a nonrelational NoSQL type of database, which means that the data is not stored via relational tables. Because of this, different NoSQL databases store data differently. Some databases store data through key values, which are used for consistent latency and performance. Some store data through graph structures, which are a type of data structure that contains nodes that connect to each other through different vertices, which may require either a depth-first or breadth-first search to traverse. MongoDB specifically stores its data through JSON documents, with a flexible API for iterative development. The documents also support embedded fields, which means that the documents can store relations and lists of data within themselves instead of external tables that all relate to each other.

Another option for us would be an SQL relational database, such as MySQL. A relational database is a database that is split into various tables, each with their own fields and parameters. These tables allow us to access data in relation to data in another portion of the database. SQL, which is short for Structured Query Language, is the language used for communication with data inside of these types of databases. Out of all of the options provided, MySQL seems like the best fit for an SQL database for this project, as it is open source and has been around since 1995, so there would be no shortage of resources or community for the database if we were to run into issues with it.

Of course, SQL and NoSQL databases each have their pros and cons to them, what kind of database that would be acceptable for the project depends on what the project needs out of the database. Here are some advantages of using a NoSQL database over an SQL Database:

- NoSQL databases have more flexible data models, which allow for changes to be made easier compared to SQL databases. A good application for this is integrating new features quicker.

- NoSQL databases have horizontal scaling, compared to SQL databases' vertical scaling. Vertical scaling consists of allocating extra CPU, memory, and disk space, which causes upgrading to be more costly due to the price of hardware, as well as causing downtime to happen more frequently when the server reaches maximum load. Horizontal scaling instead adds more machines to the database and splits the load between them. This saves money on upgrades, as instead of replacing hardware for more powerful hardware, you would instead be purchasing more of the hardware you already have to scale the server. This also allows better performance for the database and the chance of downtime is less often than of an SQL database.
- Queries are usually faster in NoSQL databases as data in NoSQL are stored to be optimized for queries, as any data that is accessed together is stored together. SQL databases have their data normalized, and queries usually require traversing multiple tables to get to a certain object. As tables grow larger and contain more objects in them, this can cause queries to take longer to go through as the database grows.

Here are the advantages of using an SQL database over a NoSQL database:

- SQL databases have more consistent data models. As NoSQL databases do not split their data into tables, this means that data that is not stored together cannot be accessed together. As opposed to SQL databases, where everything is linked together. This also helps visualize the database itself and makes sorting and linking new entries to it easier.
- Vertical Scaling is simpler compared to horizontal scaling. While horizontal scaling allows better performance by having more machines to the same server, vertical scaling has better performance simply by upgrading equipment. The increase in RAM increases performance alone, while increasing storage allows the SQL server to store larger entities than horizontal scaling. For example, images may take up too much space for a server running a NoSQL database to be able to handle.
- As SQL databases have been around longer than NoSQL databases, this means that there is much more documentation on SQL databases than NoSQL databases. On top of that, SQL databases are more beginner friendly as the languages used for them are standardized and unlike NoSQL databases, with databases being stored via json documents or nodes in a graph, SQL databases typically do not require any intermediate computer science knowledge.

Ultimately, the database that the team is utilizing will be MySQL, the SQL database. The main reason why this is the database we are utilizing is that it is the one our sponsor, Dr. Amy Giroux, is most familiar with. The ease of use compared to the familiarity would allow her to maintain and enhance the database after our project has been completed. This also allows for future proofing in case a future team is to work on a project that would require connectivity or modification on the database.

The other main reason why we will be using MySQL is that the project follows a relational model. The database contains the artifacts that are to be shown on the site, with many traits, such as people involved, the location, and images taken of it, split into smaller tables that link back to it. The specifications are in greater detail below in the Database Tables section.

## 5.17 Unit Testing Framework

Unit testing is used to validate that each component of code is performing as it should. With unit testing we can reuse code because we know it works and catch bugs earlier in our developmental process rather than at the end of the project. There are two types of Unit testing Manual and Automatic. Manual unit testing is done when you update your code and then go into the browser and refresh the page or manually enter in data to see if the change is making the app perform correctly. For automated Unit testing code is written to constantly test the function as the application is developed. This code is not included when the application is deployed, and a framework is typically used for these automated tests.

There are many different frameworks that work with JavaScript in both front and backend. Some of these frameworks would be Mocha JS, Jasmine, and Jest. Jest was made by Facebook with the purpose of testing React while Mocha and Jasmine were initially made to test Node applications making all three a good choice for this project. We decided to not go with Jasmine for our project because it is better suited to be used with Angular and it would take more work to get it running than Mocha or Jest. In the end we decided to go with Jest as it is better suited for smaller projects than Mocha and it requires less libraries to set up.

## 5.18 Enzyme

Enzyme is a Unit Testing tool that can work alongside Jest or any other testing framework within React. The main purpose of Enzyme is to make it easier to test any of React's components, enzyme is not intended to work with the back end, but only with the UI. Enzyme is installed using npm with the line npm install --save-dev enzyme enzyme-adapter-react-16 enzyme-to-json and package.json will be updated to include "jest": { "snapshotSerializers": ["enzyme-to-json/serializer"] }. Enzyme-to-json is an improved format for json comparison because it allows you to pass an Enzyme component to Jest using the line: expect(toJson(rawRenderedComponent)).toMatchSnapshot(); [34]. A Jest snapshot test will have a snapshot be created and then compared against another snapshot that is saved with the test. Enzyme-to-json minimizes the code that is needed to do this and allows the test to be much more pinpoint. By being more focused it is easier to find why you may be getting an error for a snapshot. Enzyme-to-json is a bonus and not one of the main features within Enzyme, but we do plan on using it for our project. Figure 35 is an example of a snapshot test that uses enzyme-to-json. When the test is run it will create a snapshot file that will be compared to whenever the test runs.

```
import React, {Component} from 'react';
import {shallow} from 'enzyme';
import toJson from 'enzyme-to-json';

it('renders correctly', () => {
  const wrapper = shallow(
    <MyComponent className="my-component">
      <strong>Hello World!</strong>
    </MyComponent>,
  );

  expect(toJson(wrapper)).toMatchSnapshot();
});
```

Figure 35: An Example Snapshot Test Using enzyme-to-json [34]

Now one of the main features of Enzyme is shallow rendering. The purpose of shallow rendering is to make the test simpler than the tests that are used in Jest.

The shallow component will only render one component at a time meaning it will not render a component's children. This helps isolate pieces of code and will increase the chances of finding exactly where a bug is inside of a snapshot whether that be the main component or its children. Full rendering or mount is basically the opposite of shallow rendering. Instead of only rendering a single component a whole page can be tested by rendering a web page [35]. To run this test locally and not in a browser you need to include a library named jsdom that acts similarly to a browser. With how simple the website is we ended up not implementing Enzyme as a stretch goal because it is not necessary for Unit Testing for the website.

## 5.19 System Testing Tools

System testing is used to test a fully developed product and make sure that the system is working end to end. We will use this to track data as it goes through our web app and ensure that the correct changes take place as the data flows as well that the user experience is up to standard. A system test is considered a Black box testing technique because the test does not require any information about the code. The test is going to be run in a browser to simulate real world scenarios as closely as possible.

There are many tools that we can use for system testing, but the two ones we decided to research are Cypress and Selenium. Both can be used to write end to end tests for web applications. Selenium has been around for many years and supports a lot of different languages. Meanwhile Cypress is much newer in comparison being made in 2014 and it only supports JavaScript. One big drawback of Cypress is that it only supports Google Chrome, Microsoft Edge, and Firefox while Selenium also supports Opera and Safari along with others. In the end the fact that Selenium can automate web browsers and that it supports more browsers meaning it can be tested more makes Selenium our choice for a System testing tool over Cypress.

# 6 Overall System Design

The TGP website is made up of several components which will be described in depth over the following pages. The block diagram and table below give an outline of how the entire system we are making will interact (Figure 36).. The list below also goes into every component, a short description of the component and who is responsible for it.

| Component | Leader | Description |
|---|---|---|
| Front End | Shelby and Jayden | Allow users to search artifacts, view searched artifacts on a map, and input artifacts to be sent to the admins. Admins can login and have access to the database. |
| Back End API | Tim | Receive CSV files for Zooniverse and through a REST api add, edit, delete, and search artifacts based on information from the website. |
| Leaflet Mapping | Shelby | Takes coordinates for searched for artifacts and places pins on a map for each one. Information will pop up about the artifacts when the pin is clicked. |
| MySQL Database | Alex | Database that will store all artifact information. |

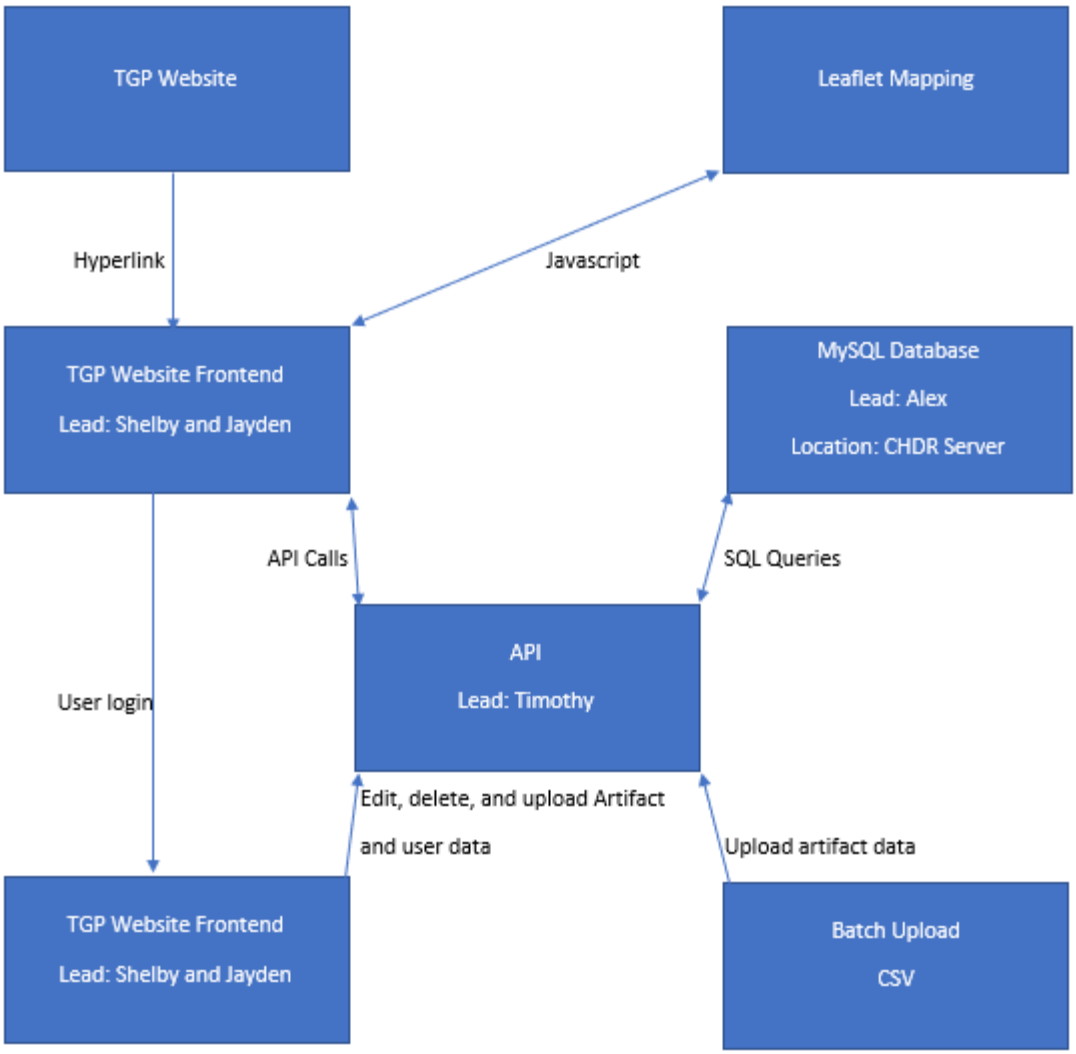


Figure 36: The Gravestone Project (TGP) Block Diagram

There is currently a website for the TGP, http://thegravestoneproject.com/. There will be a link from this current website to the new one being made. The website's front end will be in the same style as the current website and research is being done on the backend to determine which languages will be used. This website is going to be connected to an SQL database of artifacts and website users will be able to query to look through the artifacts. There will be a login for Admins so they can add, delete, or edit artifacts directly from the database. The mapping of the artifact will be done on the website through Leaflet which is a JavaScript Library. Zooniverse is going to be used for organizations to transcribe and categorize images or artifacts and from there the data will be uploaded directly into the Artifact database.

# 6.1 Web App

## 6.1.1 Design Summary

The goal of the front-end of the TGP's web application is to serve as the interface between the user and the database/functionalities of the system. This interface will be accessible from a link on The Gravestone Project's main page that will be added post-delivery of the project.

To make the site as easy to use for visitors as possible, basic (non-administrative) users will not be required to create accounts to access any website functionalities. Users, upon accessing the interface, will have the option to search keywords, names, dates and timeframes, locations, and select from various content filters or sorting types. Searching will repopulate the page with the results of the current search. Users can open up a map view on the results page, where all results on the current page are represented as pins on the map. Selecting a pin will reveal the name of the artifact, alongside any other identifying information. Selecting a result on the list will open an overlay containing all the information of that artifact.

Though basic users will currently have no incentive to create accounts, the options to login and signup are placed in a dropdown menu on the website's toolbar. Both login and signup will be overlays on the current page, rather than their own pages. Once logged in, users will be returned to the project homepage.

Administrators will have access to an administrative panel to perform system maintenance and manage users. From this panel, admins will be able to…

- Promote, demote, and delete existing users.
- Edit, add, and delete various tags used on artifacts.
- Perform singular and batch uploads of artifacts.

Upon viewing an artifact, admins will have access to Edit and Delete options on artifact to perform the respective tasks. Attempting to delete an artifact will prompt an alert box. Attempting to edit an artifact will open the same form that opens when creating a new artifact, with any existing data already entered in the respective fields. Required fields will be marked as such, and the administrator will be alerted if they attempt to submit changes having not filled out all required fields.

Basic users will not be able to submit additional artifacts, request an edit, or request a removal through the site. Users will instead be directed to a form hosted by Google Forms, where they can enter the data for an artifact, or request a different action. Administrators will review Google Forms submissions as needed.

The front-end will maintain the same stylistic choices as the current website, while incorporating design standards that make transitioning from a wireframe/prototype to a functional website during development as seamless as possible.

Below is Figure 2, a UML Use Case Diagram that visualizes the functionalities of the website, and how each type of user (basic users, and administrators) can interact with the system (The extension to the TGP website). Note that a basic user is able to sign up and sign in, but this interaction is not expressed in the UML diagram, as accounts will serve no purpose to basic users.

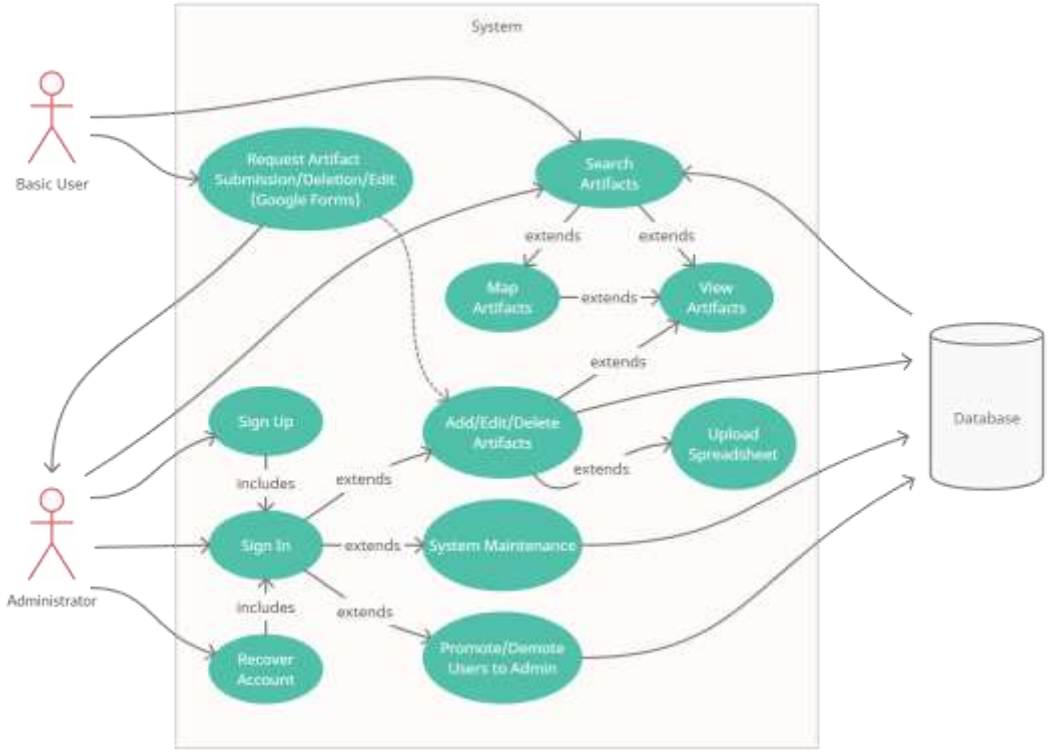


Figure 37: Website Use Case Diagram

### 6.1.2 Design References

Before the development of the front-end, a prototype/wireframe is being development in Figma. To get a better idea of the breakdown and functionality of the website, various other website's implementations of functionalities and designs were considered. The following are screenshots of the functions of other web applications that visualize the intent of the web app design.

Before considering the design solutions of other websites, we first considered the overall visual design of the existing TGP website. Figure 38 is a screenshot of the landing page of this website (thegravestoneproject.com). Through inspecting the website HTML on our browser, we want to do our best to mimic the padding, colors, font, and stylistic choices of the current website. Maintaining a mostly gray design is very important, as is continuing the use of Fira Sans being the main font used.

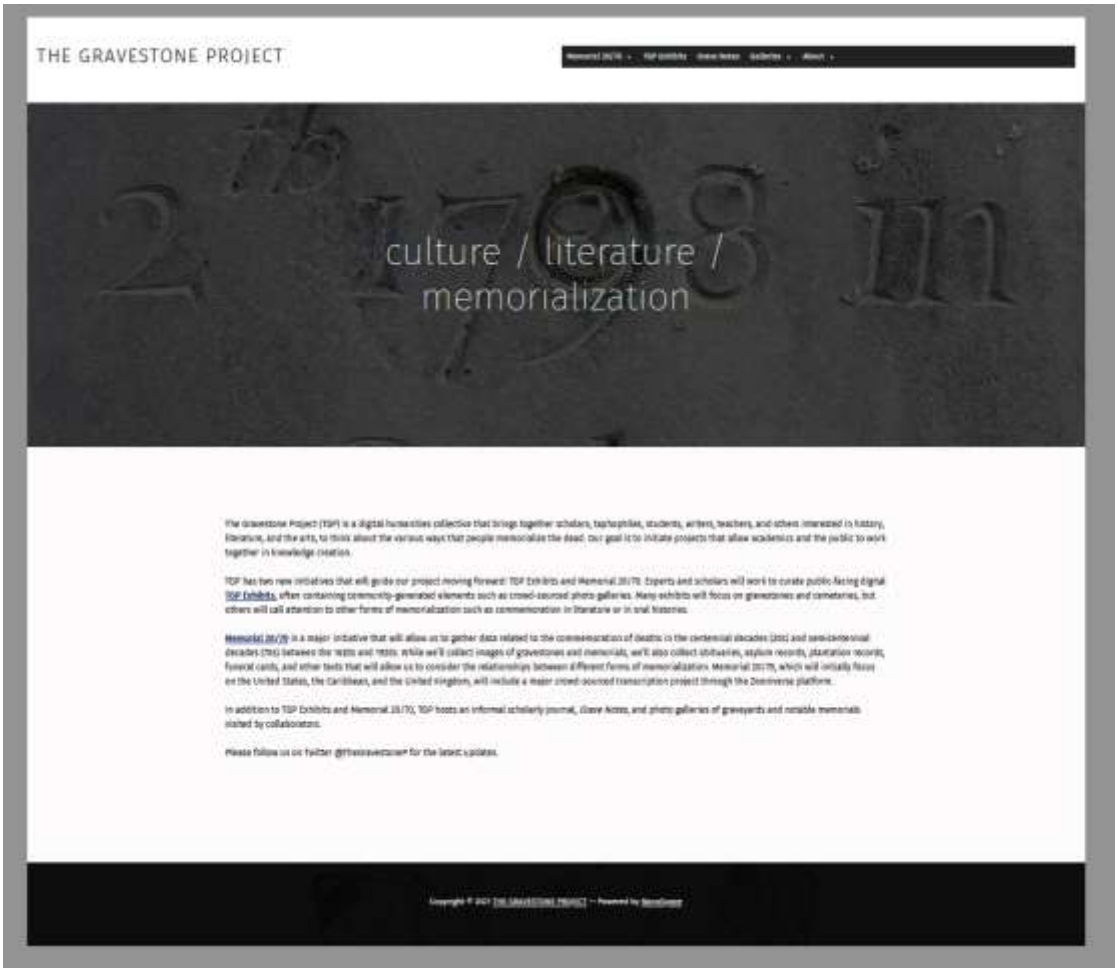


Figure 38: The Landing Page of the TGP WordPress Website [5]

Figure 39 shown below is a screenshot of a mapped Google search, illustrating similar functionality and design to what we want to achieve for the artifact mapping functionality, including:

- Labeled pins placed on a map for each result.
- Map zoomed in to the smallest area still containing every pin.
- A left side panel with a result list, broken into pages of 20 results each.
- A tab to the right of the side panel to collapse and expand it.

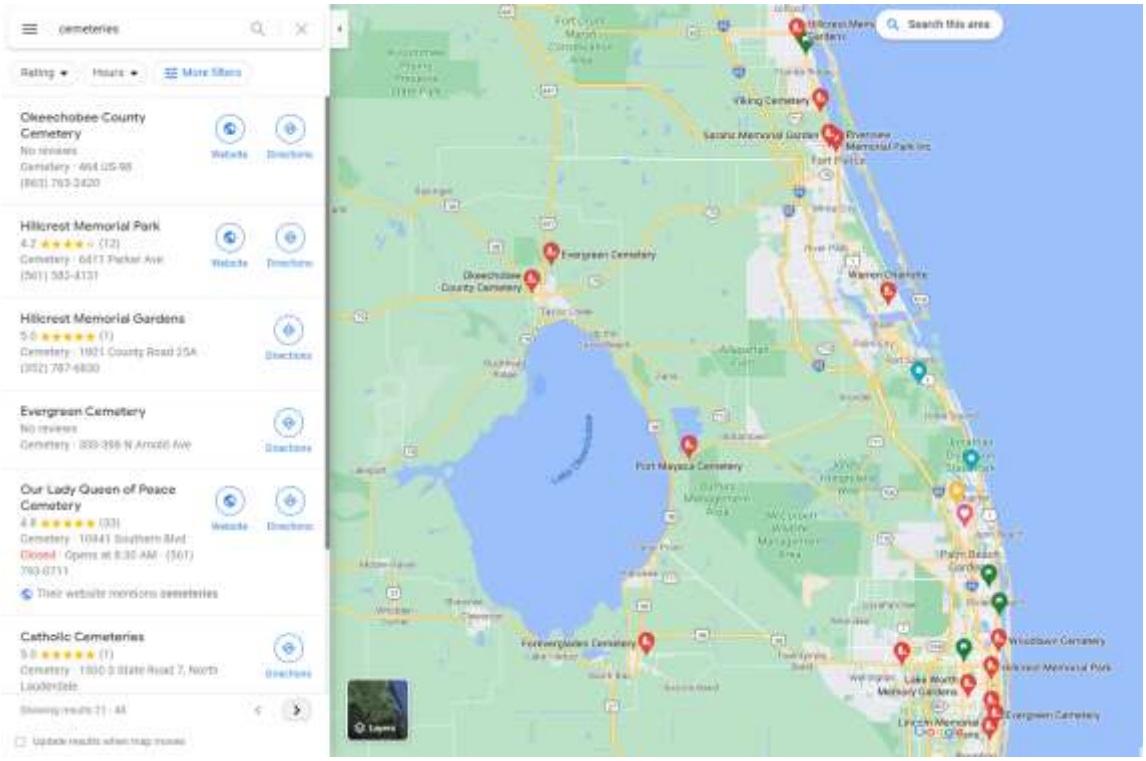


Figure 39: A screenshot of a mapped Google search for “cemeteries” [36]

Figure 40 is a screenshot from FindAGrave.com, showcasing similar data to what we will be providing to users on an item’s page. Several functions and tabs on this screenshot, including suggesting edits, adding photos, and requesting photos, will be functions within the project itself, but will not be directly handled by the website.

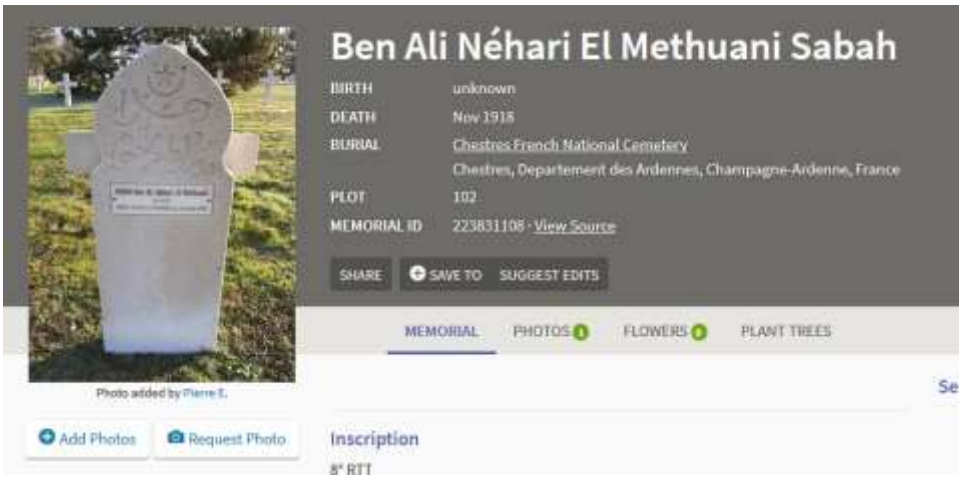


Figure 40: A gravestone's page from FindAGrave.com [37]

Figure 41 is a screenshot of the filter sidebar on Target.com, inspiring the design, spacing, and separation of the search filters that will be available on the project website. The ability to search through a specific filter type, and scroll through choices within filter type, will be implemented as well.

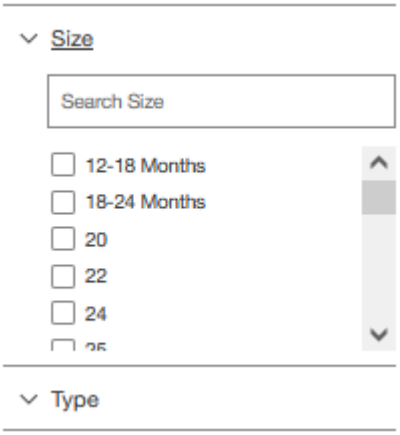


Figure 41: Example search filter design on Target.com [38]

### 6.1.3 User Interface Prototyping

In order to consolidate our understand of how the website will function and flow, we created a wireframe, or prototype, of the website. These designs are intended to reflect how the actual website will be visually constructed, how we intend to meet or use case requirements through UI.

Every component of the website was designed with the style of the current TGP website in mind. Our intent is to use this exact Figma prototype to build the front end off of, but this may see further changes as we discover new challenges or receive requests from our sponsor.


Figure 41: Additional Menu Options on TGP Website

When users first navigate to thegravestoneproject.com, they will land on the WordPress site currently being maintained. Through a menu strip on the website, users can select from two new menu options: **Search Artifacts** and **Artifact Submission**. These additions will be made by the members of the TGP that maintain the WordPress site, but will also be seen parroted in the wireframe. Figure 41 illustrates what these additional menu options will appear if a user selects **Artifact Submission**, they will be redirected to the Google Forms page to fill out their request. If a user selects **Search Artifacts**, they will be taken to the main search page, shown below in Figure 42.

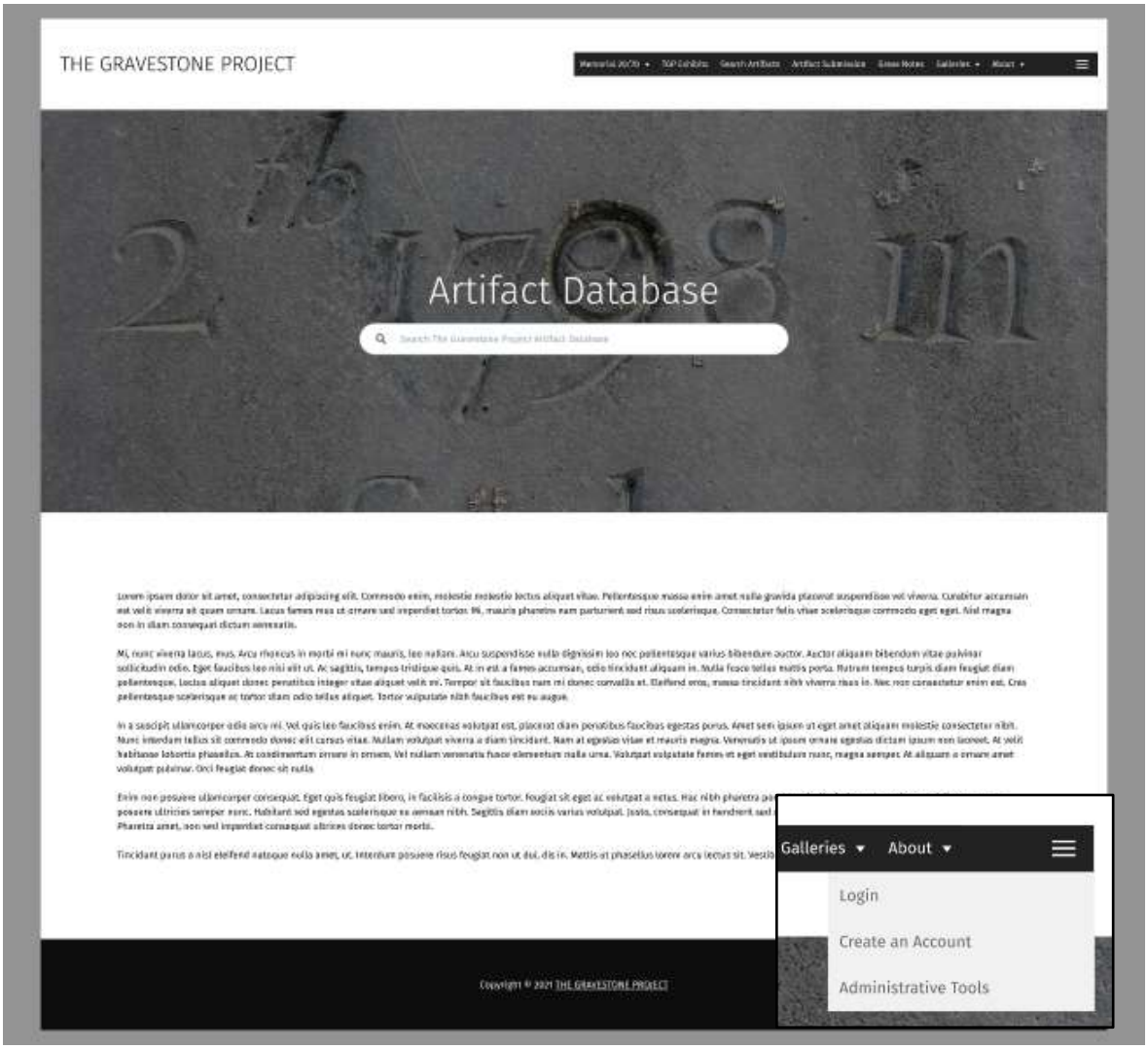


Figure 42: Wireframe of Search Page and Menu Options (bottom left)

From the main search page, users will be able to select from a hamburger menu (an icon that looks like three lines stacked horizontally) for the options to Login, Create an Account, or (if the user is already logged in), access the Administrative Panel. This menu is shown magnified in the bottom left corner of Figure 42.

Figures 43 and 44 showcase the modal designs for logging in and signing up. The user can swap between the two by selected the “Log In” and “Sign Up” hyperlinks on each modal.


Figure 43: Wireframe of Login Modal

The **Login** modal will require the user's email and password, and will display messages upon failed login attempts, and upon the locking of a user account (for too many login attempts).

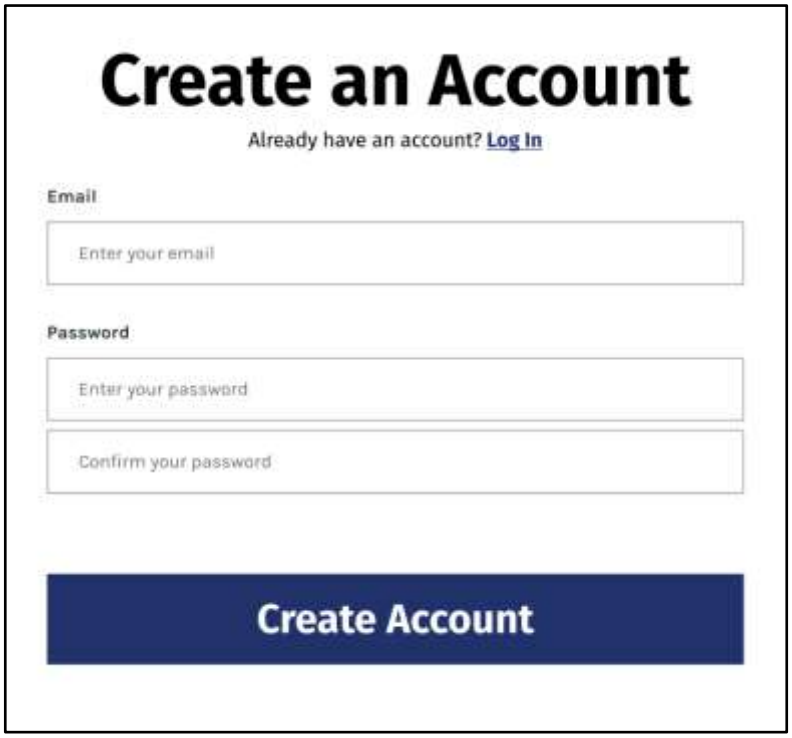


Figure 44: Wireframe of Signup Modal

The **Create and Account** modal requires the user to provide an unregistered email address, and to input their password twice. Messages will appear to inform the user of mismatched passwords, passwords not meeting requirements, or the attempted use of an email that is already registered on the site. When either modal is open, they will appear as overlays on the center of the screen, and cause the page in the background to darken. Uses can exit these modals by clicking anywhere outside of them, or by successfully submitting the modal form.


Figure 45: Wireframe of Email Verification Success Page

When a user signs up, they will receive an email notifying them to verify their own email. Upon clicking the provided link, the user will be sent to the page shown in Figure 45. From here the user can access the basic toolbar present on most pages, or simply press the button to bring them to the search homepage.

There will also be a variant of this page in the case that an error occurred when the user attempted to verify their email. The user will receive the message titled “Unable to Verify” with the content “An error occurred while verifying your email. Please try again in a few minutes.” with the button to return to the search homepage remaining as pictured.

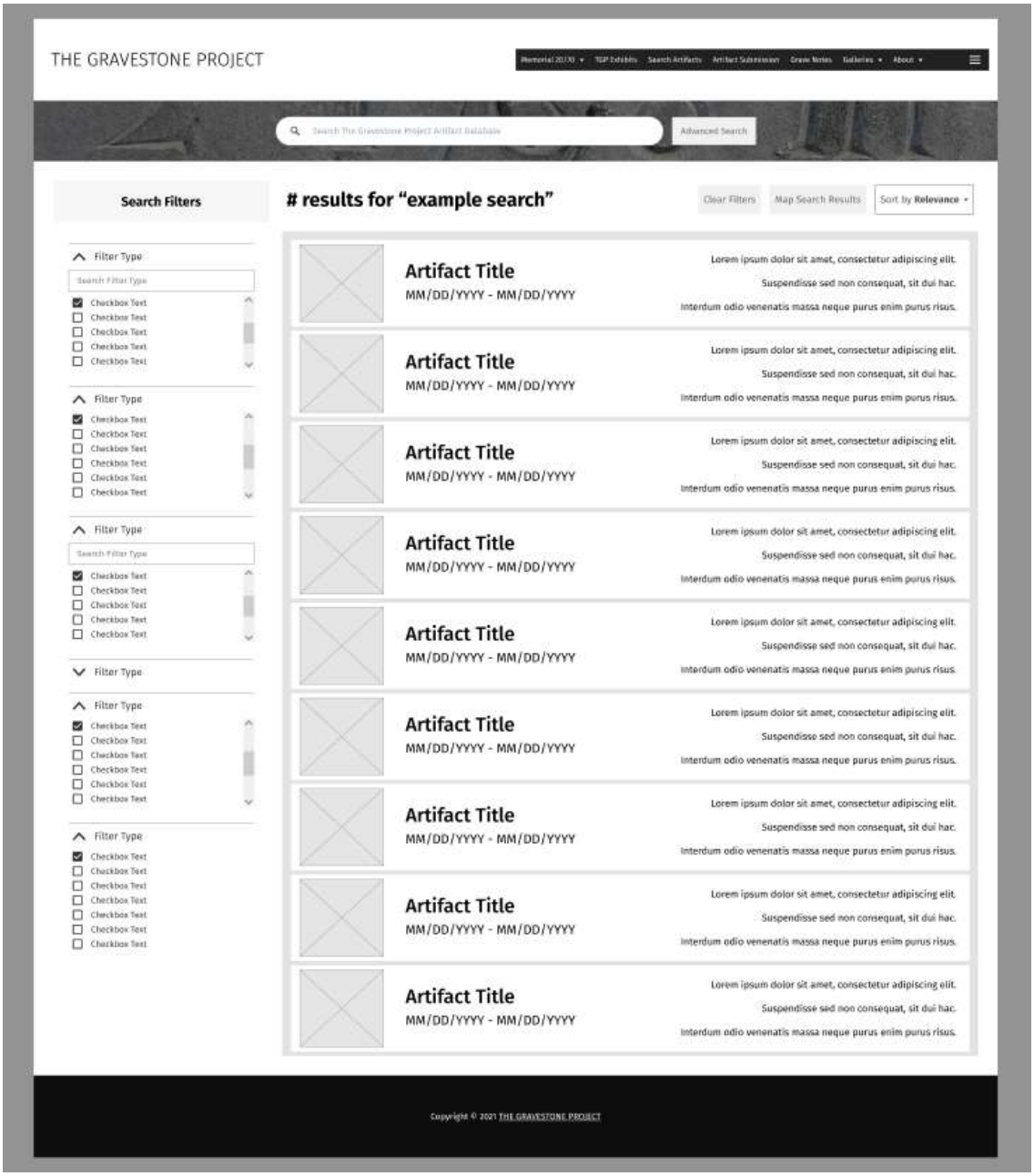


Figure 46: Wireframe of the Search Result Page

Once the user submits a search for an artifact, they will be redirected to the results page pictured in Figure 46. All results for the search will appear as rows, each showing a thumbnail of the artifact's featured picture, the artifacts title, relevant dates, along with several other fields that are left-aligned. From here, the user can swap the view to an advanced search to narrow their search with specific fields. They may also filter out content using the filters placed on the left side of the results, clear their currently applied filters, and change how the results are sorted.

If the user is logged in under a current administrator account, an additional button will be available on the page for the administrator to **Submit an Artifact**. This will open up a modal form where the admin can fill out the known fields of the artifact, and upload it directly to the database, rather than requesting it through Google Forms as a non-admin user would have to do.


Figure 47: Wireframe of the Map Result Page

If the user selected the **Map Results** button located on the search results page, they will redirect to the page shown in Figure 47. On this page, all results for a search are placed as pins on a map that will initially be zoomed into the smallest area allowable that still shows every result. Users will be able to use their mouse to navigate the map, and their mouse wheel to scroll in and out.

A single artifact may have multiple pins placed on the map if there are several locations tied to different significant dates. For example, “John Smith’s Gravestone” may have a blue pin locating John Smith’s grave, a red pin locating John Smith’s birthplace, and a yellow pin locating John Smith’s location when he passed. Using the legend toggle on the top right of the page, a user can choose to show any combination of these three location types on the map.

On the left of the map results page is a collapsible sidebar listing the results of the current search in a compressed design. Results on the sidebar will appear in pages of 20, which the user can flip between. Regardless of the page a user is currently

on, pins for all results will appear on the map. Additionally, the user can choose to swap the sidebar to fill out an advanced search, or to toggle the content filters of the current search.

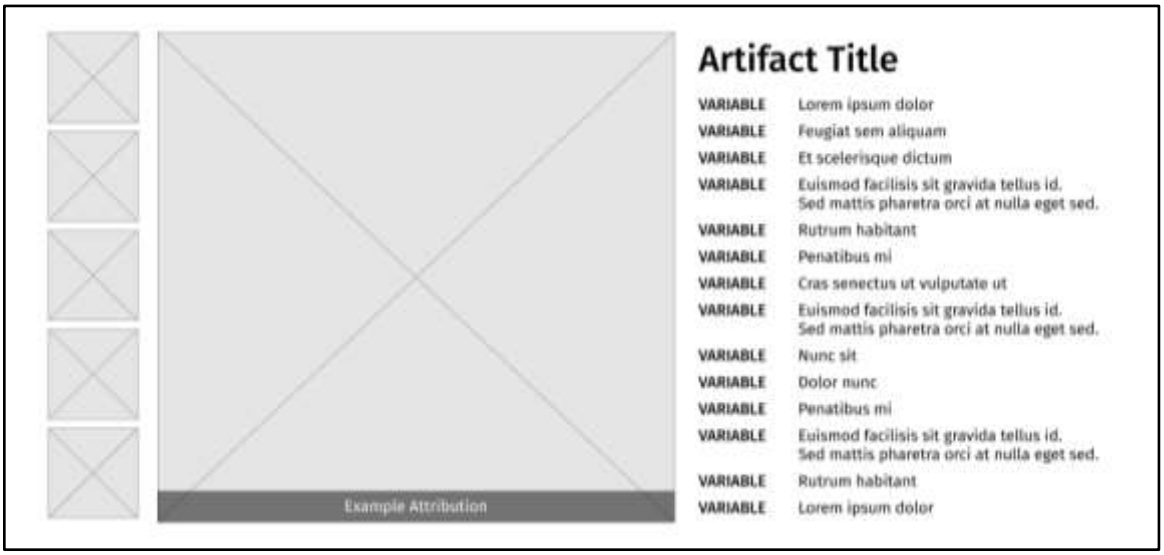

Figure 48: Wireframe of an Artifact Modal

When the user selects a search result on either the Search Result page, or the Map Result sidebar, a modal will open up on the screen to show the full data of the artifact, as seen in Figure 48. The very left of the modal will have a list of photograph thumbnails to select from--selecting one will bring the photograph into the larger box, and will display the photograph attributions at the bottom. If an artifact has more than five images, the user will be able to scroll infinitely through thumbnails.

In the case that the user is currently signed in as an administrator, additional buttons will appear on the modal to edit and delete the artifact. Choosing to edit the artifact will open the same modal as used for submissions. However, instead of the modal containing empty fields, it will have all of the artifacts current data filled out in the respective fields. Choosing to delete an artifact will open an alert box to warn the administrator that they are about to delete an artifact, and that this process cannot be undone. Choosing to confirm the delete will close both the alert box, and the artifact’s modal.


Figure 49: Wireframe of The Administrative Panel Page

Shown previously in Figure 42, administrators can access the admin panel through the hamburger icon on the site's menu bar. When selecting it, the admin will be directed to the Administrative Panel page, shown in Figure 49. At this time, admins will be able to access several tools within this single page:

- Selecting **Artifact Submission (Single)** opens a tool on the left side of the page to fill out the artifact submission form embedded into the webpage, rather than as a modal.
- Selecting **Artifact Submission (Batch)** opens the tool for admins to upload spreadsheet-type files (CSV, Excel) to automatically load the artifact data onto the database. After the upload is complete, the admin has the choice to review the newly created artifacts one by one for a chance to edit them.
- Selecting **User Accounts** opens the tool for admins to view all accounts created. At first, all accounts will be listed, and admins can perform a search to narrow down the list. Admins can also toggle to only show other admins, or to only show non-admins. Each user will appear as a separate column listing their email and userID, alongside buttons to promote the account to administrator, demote the account from administrator, or delete the account entirely.

- Selecting **Toolbar Manager** will open the tool for admins to add, remove, rearrange, and edit menu items on the top left toolbar. Admins can convert a menu item between a singular item and a dropdown menu, construct dropdown menus, and edit menu item links.
- Selecting **Featured Image** will open the tool for admins to change the image featured on several pages of the site. This is the image acting as the background to the search bar on both the main search page and the search results page.
- Selecting **Tag Manager** will open a tool for admins to view all existing tags, edit them, delete them, or add new tags. Attempting to delete a tag will open an alert box seeking confirmation of deletion.

### 6.1.4 User Accessibility

Alongside the main purpose of the site, our goal is to create components and visual aids that uphold basic user accessibility standards. This means most components (such as buttons and text fields) are created with both color-based and shape/icon/text-based visual aids. There are a few components within the design that do not uphold these standards, as our primary goal is to copy over the style of the existing website.


Figure 50: Color Variants of Buttons

Button variants are one such acception to the rule of accompanying a shape-based visual aid with a color-based one, since we had to match these with the existing site. Still, the button component has several visual properties:

| **Property** | Color | Dropdown Toggle | Hover Toggle |
|---|---|---|---|
| **Variant** | Light | No Dropdown | Not Hovering |
| | Dark | Dropdown | Hovering |

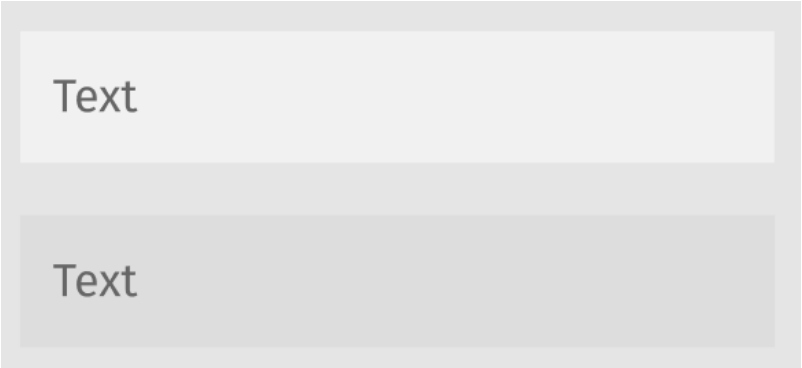


Figure 51: Color Variants of Dropdown Menu Buttons

Additionally, if a button is dropdown enabled, its dropdown items have the “Hover Toggle” property as well, with variants for when the item is hovered over vs. when it is not.

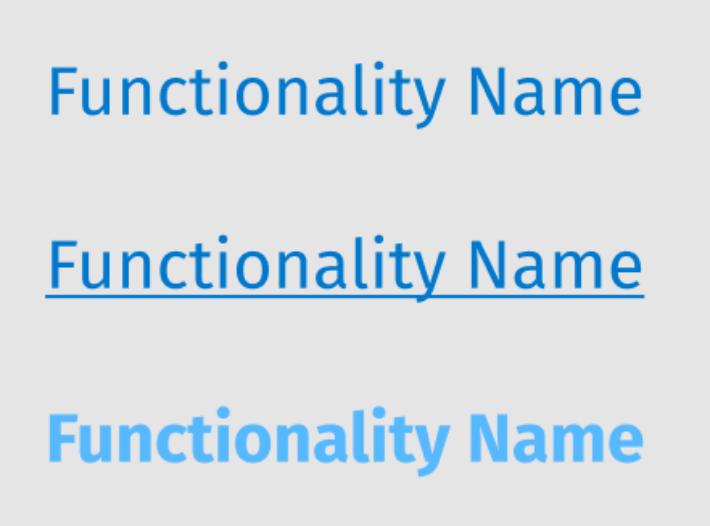


Figure 52: Color Variants of Admin Panel Tools

Administrative panel tools showcase the consideration of shape and color between its interactive variants. When not selected and not hovered over, the tool text appears like the top variant in figure 52, with basic text. When the tool text is not selected, but is hovered over, it will appear like the middle variant, with underlined text. When a tool text is selected, it will appear like the bottom variant, with a lighter text color and bolded font.

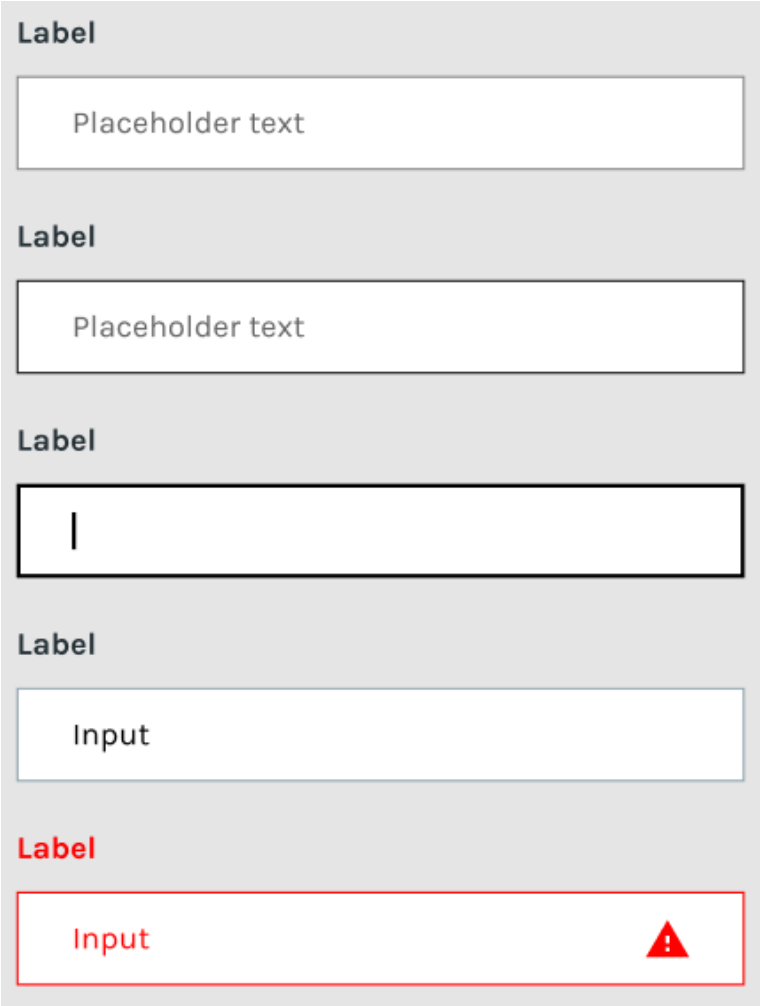


Figure 53: Property Variants of Text Field

Input fields were designed with the multiple visual aids as well. The top variant is the default text field, and the second variant (with a darker online) is caused when the text field is hovered over. The middle variant occurs when the text field is selected, and causes the outline to darken and thicken. The four variant appears when the textbox is unselected, but contains input. And finally, the fifth variant at the bottom indicates that there's an error with the input in the text field. This variant

will proc when require fields are left blank, or when a user runs into an error during login or signup.

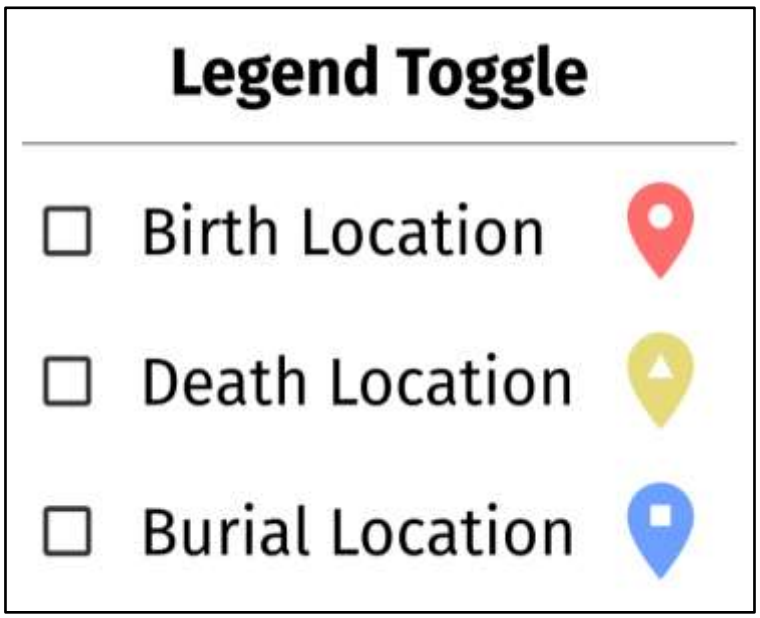


Figure 54: Closeup of the Legend Toggle from Map Results

On the map results page of the website, each artifact has at least one pin representing it on the map, and any one pin can refer to a Birth Location, Death Location, or Burial Location associated with an artifact. To differentiate these on the map, each type of location has an assigned color and an assigned pinhole shape to help with color-based and shape-based visual differentiation.

### 6.1.5 Library Usage Overview

All of the libraries and frameworks that will be used to develop the front-end of the website are listed below. Each component falls under either a free standard license or the MIT License, both allowing non-profit commercial use [39, 40, 41].

- React.js (JavaScript Library, Web App Framework)
- Material-UI (React Framework, UI Design Language)
- React Bootstrap (React Framework, UI Component Library)
- styled-components (React/Javascript Library)
- Leaflet (Javascript Library for Interactive Maps)

- React-Leaflet (React Library, Map Components for Leaflet)

### 6.1.6 React Components

Each react component will be initially built under a test domain, and have URL extensions that best describe their purpose.

- **Global-Usage Components** - Parts of the website that appear on every page, or almost every page.
    - WebsiteHeaderToolbar - Website header containing website title and toolbar with dropdown menus and links to different pages.
    - WebsiteFooter - A general footer with the copyright for the website.
    - LoginForm - Fields for login modal
    - SignupForm - Fields for signup modal
    - ForgotPasswordForm - Fields for forgot password modal
- **SearchResultsPage (URL: “/search”)** - Accessed by users by selecting the “Search Artifacts” button on the toolbar.
    - SearchForm - Fields for users to input search query
    - Map - Collapsable map that renders pins of various location types
    - SearchFilters - Fields for users to input content filters
    - SearchResultOptions - Menu for users to sort content
    - SearchResults - List of query results
    - Result - Individual result of a query
    - WebsiteSearchResults - Parent component to above components
- **AdminPage (URL: “/admin-panel”)** - Accessed only by logged in admin-level users by selecting “Administrative Tools” from the Account dropdown menu on the toolbar.
    - AdminToolForm - Menu of the various admin tools an admin can view/use
    - ArtifactSubmissionBatch - Tool to submit multiple artifacts via CSV
    - ArtifactSubmissionSingle - Tool to submit a single artifact
    - TagManager - Tool to perform CRUD operations with tags

    - UserAccounts - Tool to moderate and change permissions of users
- **EmailVerificationPage (URL: “/email-verification”)** - Accessed by the user after signing up on the website.
- **PasswordResetPage (URL: “/reset-password”)** - Accessed by the user after clicking the password reset link sent to their email upon filling out the “Forgot Password” form.
    - PasswordResetForm - Form for the user to change their password
- **SuccessfulPasswordResetPage (URL: “/successful-passwordreset”)** - Accessed by the user after successfully resetting their password on the PasswordResetPage.
- **SuccessfulSignupPage (URL: “/successful-signup”)** - Accessed by the user after clicking the email verification link send to the email they signed up with.
- **AccessDeniedPage (URL: “/admin-panel” when not logged in as admin)** - Accessed by the user when attempting to view the AdminPage without being logged in with an admin-level account.
- **NotFoundPage (URL: “/<any invalid address>”)** - Accessed by the user when attempting to visit an extension of the website not listed above.

## 6.2 Database

### 6.2.1 Design Summary

The database used for this project is a MySQL database provided by our sponsor, Giroux. As the main purpose of the project is to log artifacts and provide information about them to the public, a main database would be important to store all of the relevant data. Said data would include the artifacts, the people pertaining to them, and the locations that host them.

The schema consists of 9 main tables: attributions, users, submitters, locations, people, collections, images, artifacts, and tags. Alongside those, there will also be two sub tables made to link tags to the main tables. The SQL nature of this database allows for easy and intricate connectivity between each table in the space, as shown in the ER (Entity Relationship) diagram in Figure 54.

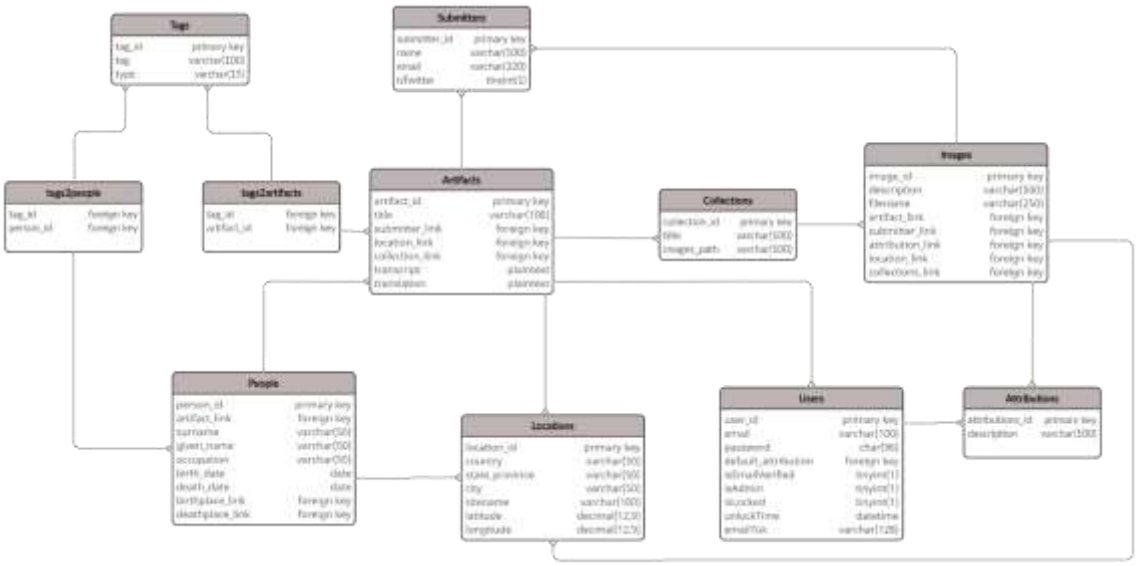

Figure 54: ER Diagram of the Database

There are two different tiers of database interaction: the way that users and guests of the site interact with the database, and the way that site admins interact with the database.

First, User Interactions. The average user of the site will be able to search for specific artifacts, as well as different people or locations that relate to them. By

default, the artifacts may be listed in alphabetical order, but there will be a search bar that will utilize a query filter, causing only certain portions of the database to be listed. This topic is documented further in section 6.2.2.

A proposed feature of the site is to allow users to “map out” the artifacts. The way this would be done would be to be familiar with a map application’s widget and API, such as Google Maps. A query would be sent to the database to retrieve the latitude and longitude data from the location data, which would be fed into a mapping function linked to the map widget, in which the location would be pinned on the map. Clicking on each pin would send another query, this time to retrieve data on the artifact that is located at the pin.

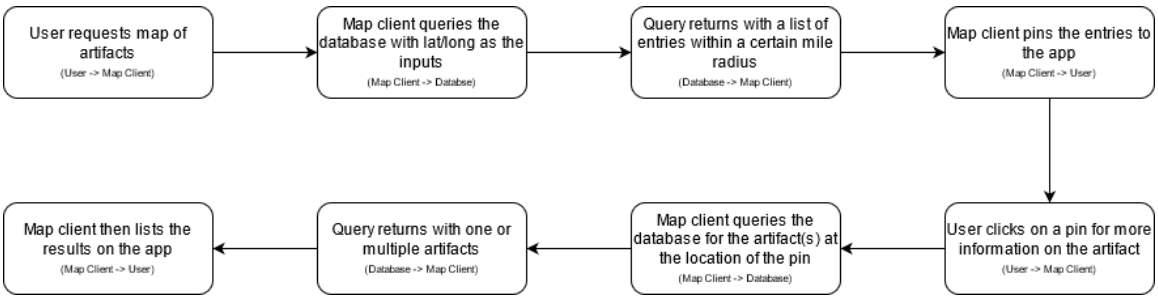

Figure 55: Diagram of Map Widget and Database Interaction

It is important to note that site users can only send out queries to retrieve data, nothing that the user can do can modify the database in any way, this includes submitting artifacts and requesting modifications to existing artifacts. The database does not even store that data as temporary entities, instead, for actions like those we will be utilizing Google Forms, which uses Google’s Bigtable NoSQL database instead of our MySQL database. This is to make it easier for the admins to review submissions, and accept or deny them accordingly. Utilizing Google Forms instead of letting the users having access to directly add entries to the database makes our database more secure by restricting access, reduces the chance of clutter filling up the database, and allows users to interact with the site admins via a familiar and intuitive UI.

Next, Site Administrator Interactions. Unlike the users, the site admins will have access to queries that edit, modify, and append data to the database. They will also have access to the bank of forms that have been submitted by the users of the site that contain submissions or edit requests. Upon reviewing the forms, the admins will be able to accept or deny the requests. If a form gets denied, they are deleted off of Google’s own database before they get a chance to cause ours to

be edited or added on to. If an artifact submission gets accepted, then the admin will create an append query to add data from the form to a new entry in the database. It will create a new entry in the artifacts, people, and locations tables, alongside possible entries in other tables such as images/collections if images are included, as well as the users table if it is by a new submitter. If a request to edit has been accepted, then the field with the edit suggestion would be reflected across the database by using a modify query to edit a certain field in the database accordingly.

As both admins and registered users may have an account, this would mean any sort of query for account management would be a necessity. Upon registering for a new account, the user would be prompted to enter their email and desired password, presumably twice for verification. Upon creating the account, the site would first request a query to search for an account that may have the email tied to it. If a non-NULL entry is returned, then that means an account with the email already exists and the site would prevent the user from registering for an account with a repeat password. Otherwise, if the query does not return with an existing entry, and the password meets the criteria, then an append query is requested to the users table, with the email and hashed password for security. If the system notices multiple failed attempts to log into an account at once, then it may send out a modify query to set the isLocked flag to true for that account, which will cause it to lock up for a certain period of time, preventing access to it. If, At any point, the user cannot access their account, either via a compromise leading to a lock or simply forgetting their password, they have access to account recovery, which would use a SELECT statement to retrieve their account, and then a modify query to replace the old password with a hash of the new one. The users may also edit the fields of their account, in case they change emails or want a new password. This would also cause another use of the modify query to reflect their changes to the database. The head admin may also promote average users to admins if they deem trustworthy enough. Doing this would require sending a modify query to set a certain user’s isAdmin flag to true.

### 6.2.2 Database Tables

This section covers all of the tables that make up the database.

The tables **arttype** and **persontype** are two fairly simple tables. They are small tables both primarily used to categorize artifacts or people, respectively. For **arttype**, this would include categories such as a grave or tombstone, and would

link to the **artifacts** table. Meanwhile, for **persontype**, categories would include“father”, “daughter”, “friend", “lover”, and would link to the **people** table.

It is important to note why these tables exist. It is possible that we could implement **arttype** and **persontype** as just values for the greater **artifacts** and **people** tables, but for a relational database, like the one we’re using for this project, categorical data is usually put into child tables. This way, new types of “artifacts” and “people” can be added as we come across them.

| **Table: arttype** | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| arttype_id | primary key | This field is a unique alphanumeric identifier for the entry in this table. |
| Artifact_type | varchar(50) | This field is the name of the type of artifact. An example would be “gravestone” |

| **Table: persontype** | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| persontype_id | primary key | This field is a unique alphanumeric identifier for the entry in this table. |
| Person_type | varchar(50) | This field is the name of a type of person, related to a subject. Examples include “decedent”, “mother”, “son” |

**Important Note:**
As work was progressing on the project, the **arttype** and **persontype** tables were **ultimately scrapped** due to their redundancy with the **tags** table, so these tables have been deleted off of the database. **Trying to call these tables will cause an error because they do not exist in the database, and their functionality has been ported to the tags table.**

As users submit images, they may want to be credited for the image. The **attributions** table will store these different attributions, and each user will contain a **default attribution** in their entry in the database. Aside from being credited by

the author, some other **attributions** include “None Required” (for if the user wants to be kept anonymous) as well as different attributions for the different types of Creative Commons licenses.

The attributions table will also feature a maintenance screen that will allow admins to add, edit, and delete contents or entries of the table. The users table will link to this table for reasons stated above, while the images table will also link to the attributions to assign credit to the image if needed.

| **Table: attributions** | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| attributions_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| Description | varchar(100) | A description of the attribution. Examples include “No attribution required” and “CC-Share Alike” |

The **users** table will consist of typical login and login-relevant information pertaining to the user. The average user will not have direct access to edit the contents of the database. Instead, they will be able to access contents of the database to search for, view, and map out artifacts in a map. They can request an artifact to be edited if it has any inconsistencies, or submit an artifact to be added to the database.

The standard account information such as **email** and **password** are parameters in this table. Alongside those, there is also an **isEmailVerified** parameter that checks to see if the user has verified their email, as well as an **isLocked** parameter that flips whenever the user exceeds their login attempts, locking their account. If **isLocked** is true, then whenever the user tries to log back in, the site checks to see if the current time is past the **unlockTime** that is logged in the table before deciding to unlock their account. There are currently no plans to give users special access that guests do not already have, but it is important to keep a generic users table as opposed to a specific admins table for future-proofing the site in case it is desirable to add in special user features in the future.

If the **isAdmin** parameter is set to true, then the user is classified as an **admin** in the database. They would have direct access to it as well as other administrative

features. They are able to view the artifacts and edit requests that the submitter has submitted, and then deny or accept them. If the request is approved by the admin, then it gets reflected in the database under the **artifacts** table. The admins are also able to perform system maintenance or account recovery if ever needed.

| **Table: users** | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| user_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| email | varchar | The email address of the user |
| password | varchar | The password of the user |
| default_attribution | foreign key | The default attribution to the user. This key is required. |
| isEmailVerified | tinyint | The Boolean field indicating if the user has verified their email |
| isAdmin | tinyint | The Boolean field indicating if the user is an admin or not |
| isLocked | tinyint | The Boolean field indicating if the account is locked or not. If set to true, then the account cannot be logged into for a set period of time |
| unlockTime | datetime | The date and time when the account unlocks if isLocked = true |
| emailTok | varchar | Email token, used for sendgrid API |

The **submitters** table stores the relevant data pertaining to the guests or users who submit artifacts to the site. Unlike the fields in the users table, which hold account information through a registration page, the information in this table will be provided by people submitting data through the submissions form. Keeping this table separate from the users table means guests could choose not to create an account, and submit an artifact anyways while not being stored awkwardly in the users table.

| Table: submitters | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| submitter_id | primary key | This field is a unique alphanumeric identifier for the entry in this table. |
| name | varchar | The name of the person who submitted the artifact. If isTwitter is set to true, then this is the submitter's twitter handle instead. |
| email | varchar | The email of the submitter |
| isTwitter | tinyint | A Boolean field indicating that the submitter wants to use their twitter handle instead of their name |

The **locations** table and **people** table are pretty self-explanatory. The **locations** table would link to the images table, where it would describe where the image was taken, as well as the **artifacts** table, where it contains the location of where the artifact is. It is important to note that we are not using zip codes for the **locations** table, as the **latitude** and **longitude** values fulfill the purpose of why we would want to implement a zip code much better. The **people** table would also be linked to the **artifacts** table, where it contains the subject or subjects of the artifact and/or whoever requested the artifact to be erected.

| Table: locations | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| location_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| Country | varchar(50) | The country of the location. |
| State_province | varchar(50) | The state or province, depending on the country, of the location. |
| City | varchar(50) | The City of the location |
| Sitename | varchar(100) | The name of the type of site at this location. Examples include "repository" and "cemetery" |

| | | |
|---|---|---|
| Latitude | decimal | The latitude of the location |
| Longitude | decimal | The longitude of the location |

| **Table: people** | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| person_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| Artifact_link | foreign key | The artifact linked to this given person. This key is required. |
| Surname | varchar(50) | The surname of the given person |
| Given_Name | varchar(50) | The given name of the person |
| Occupation | varchar(50) | The occupation of the given person |
| Birth_date | date | The person's date of birth |
| Death_date | date | The date in which the given person has perished |
| Birthplace_link | foreign key | The location in which this person was born |
| Deathplace_link | foreign key | The location in which this person perished |
| Cause_of_Death | varchar(100) | The way this person has perished. Examples include "Heart Attack" or "Blunt Trauma" |

The **images** table is the table where images are "stored". Images themselves are not stored in the database itself, but rather in an external image storing directory. Instead of directly storing the images, the database stores the file path and name that the image resides in. Image metadata, such as the subject, location, attribution, and submitter are also stored in the database.

The database stores images a bit differently depending on if it is a single image upload or a multiple image upload. If multiple images are uploaded at a time, then they are grouped internally in a collection, which is being stored by the **collections** table. Each image in the collection is renamed to **tpg-<collectionID>-<artifactID>-<sequential number>**, and is stored in a unique directory

**\images\<collectionname>**. Individual images are all stored in a common file path **\images\individual**.

| Table: collections | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| collection_id | primary key | This field is an unique alphanumeric identifier for the entry in this table.identifier. |
| title | varchar(100) | The title of this collection |
| Images_path | varchar | The directory containing all of the images in this collection |

| Table: images | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| image_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| description | varchar(100) | A description of the image |
| Filename | varchar | The filename of the image. If the image is brought via a spreadsheet import, it will be renamed to tgp-<collectionID>-<artifactID>-<sequential number> |
| Artifact_link | foreign key | The artifact that is the subject of the image |
| Submitter_link | foreign key | The user who submitted the image |
| Attribution_link | foreign key | The attribution tied to the image |
| Location_link | foreign key | The location where the image was taken |
| Collections_link | foreign key | This key is not required in a single image upload. It is required if multiple images are uploaded at once, in which this key links to the specific file directory this image belongs to. |

The **artifacts** table is the backbone of the whole database, and what most of the other tables link to. All of the data in this table is split off into separate smaller

tables. This is to better categorize various features of the artifacts, as users may be able to search for the artifacts by people or locations. The location parameter is especially useful for the mapping feature of the site, which is explained in further detail in 6.2.1. The artifact may also contain a **transcription**, which is anything that is transcribed into the artifact itself. This includes a quote on a headstone, for example. If the transcription is not originally in English, then the English **translation** is stored as a separate value in the table.

| **Table: artifacts** | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| artifact_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| title | varchar(100) | The title, or name, of the artifact |
| Submitter_link | foreign key | The user that submitted this given artifact. This key is not required. |
| Person_link | foreign key | The person relating to the artifact. This key is required. |
| Location_link | foreign key | The location in which this artifact resides. This key is required. |
| Collection_link | foreign key | The collection containing all relevant images of the artifact. This key is not required. |
| Transcript | plain text | The transcript that the artifact may contain |
| Translation | plain text | A translation of the transcript |

The **tags** table is used for the site's tagging system. These include descriptors for each artifact and person in the database. Since each tag may link to multiple subjects, and each subject may link to multiple tags, they are not able to link to each other directly in a MySQL database. As a workaround to this, there are two helper database tables to help link each tag to their subject: **tags2people** and **tags2artifacts**. Both of which only contain foreign keys, one to the tag's id and one to the person's or artifact's id respectively. As **arttype** and **persontype** have been scrapped, **tag** has a **type** field to specify if the tag is meant for an artifact or

person, respectively. This is to help with uploading batches of artifacts and to search for artifacts. For more detail of the tagging system itself, see section 6.2.3.

| Table: tags | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| tag_id | primary key | This field is an unique alphanumeric identifier for the entry in this table. |
| tag | varchar | The tag, to be used for artifacts, images, people etc. |
| type | varchar | The type of tag, examples are “artifact”, “person”, “location”, and “image |

| Table: tags2people | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| tag_id | foreign key | This field links back to the parent tag table. |
| person_id | foreign key | The person this tag applies to |

| Table: tags2artifacts | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| tag_id | foreign key | This field links back to the parent tag table. |
| Artifact_id | foreign key | The artifact this tag applies to |

Additionally, there are two extra helper tables for the tag table that are currently unimplemented. The **tags2images** and **tags2locations** are set up, but are currently unused by any of the API endpoints. They were tables Giroux wanted to be implemented initially, but we were unable to implement the tagging system for images or locations in time. They are still in the database in case another team may want to implement the tagging system in the future. Structurally, they are nearly identical to the **tags2people** and **tags2artifacts** tables above, but with images and locations, respectively.

| Table: tags2images | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| tag_id | foreign key | This field links back to the parent tag table. |
| image_id | foreign key | The image this tag applies to |

| Table: tags2artifacts | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| tag_id | foreign key | This field links back to the parent tag table. |
| location_id | foreign key | The location this tag applies to |

### 6.2.3 Search Queries and the Tagging System

Most interactions between the user and database would be through the artifact search feature of the site. The user could search for either people, artifacts, or places. In either case, they would be able to choose what they're searching for, and then write the name or title of the subject, which would then utilize a query filter. The query filter essentially reads through the entire table it is searching through, and then compares the names or titles of the subject with the string that the user inputs, and if the name or title contains the string, then it gets listed on the results.

Along with the standard search function, the site will also have a system for storing and searching for tags. The tag tables will contain various tags meant to categorize the subjects of each table they are applied to. This is useful for users who want to look for specific types of artifacts, or if they do not have a name in mind. Each subject that may have tags will have different categories of tags to choose from, for example:

- Tags for an artifact may include the material the artifact is made of, the time period of the artifact, the size of the artifact, or any extra details that the artifact may have.
- Tags for a person may include the time period the person lived in, race and ethnicity, gender, social or political status, or age range at the time of death

- Tags for a location may include urban or rural areas, historical significance, cultural significance, public or private areas, altitude, or climate. **(Currently unimplemented)**
- Tags for an image may include whether or not it is for any of the three subjects above, time period of when the image was taken, or photo orientation. **(Currently unimplemented)**

Like the normal search functionality above, the tagging system will also be executed through query filters. Our sponsor, Dr Giroux, gave us some insight on how a previous team did a similar tag search function below:

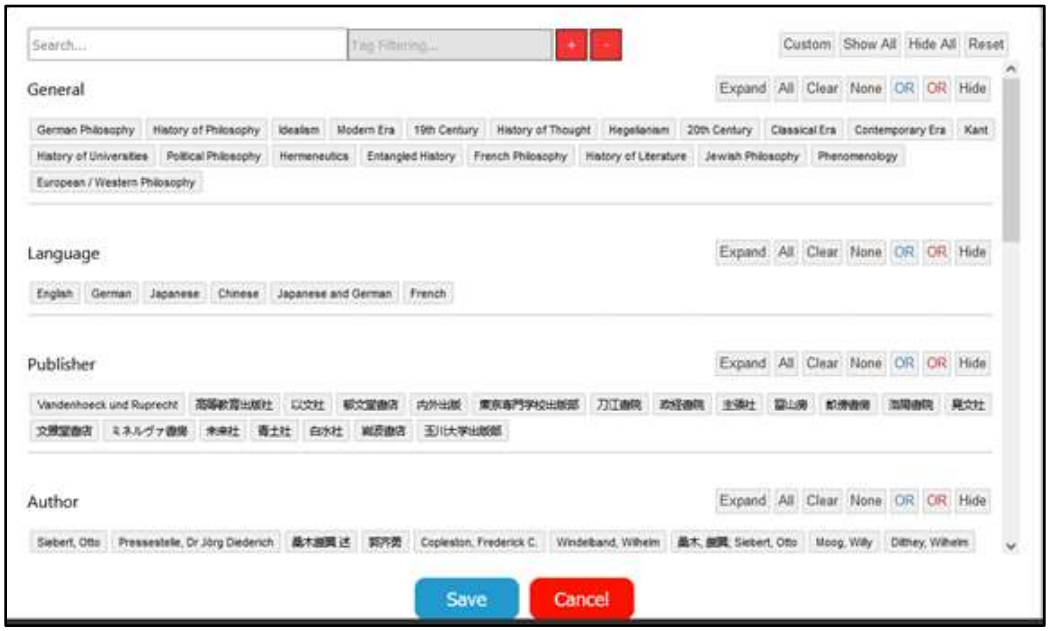

Figure 56: A Tagging System Example Provided by Dr. Giroux

We will be utilizing a similar method. Users may include tags to their search, which then will be applied to the query filter as it searches the database table. Unlike the other tables in the database, the tags2_ tables are merely acting as bridges between tags and their subjects. This is because relational databases are strictly designed to store one value per field, and do not support arrays or lists. To circumvent this, each tags2_ table links to two ids: the id of the tag and the id of the subject of the tag. For example, if we had JFK in the database with a person id of 7 and a tag “US President” with a tag id of 19, the tags2people entry linking the two will just have the respective fields of 7 and 19 only. If we add a tag highlighting his political party, with “Democrat” having a tag id of 23, then the tags2people table will add another entry on top of the previous one with respective fields 7 and 23. If then we add another person, Ronald Reagan, who would share

the “US President" tag, but would instead have a different tag highlighting his political party: “Republican” with id of 24, then the same tags2people table would add two new entries on top of the previous ones for JFK with their respective fields. This means that for each search that a query filter performs on a database will include searching through the suitable tags2_ table for a tag that is also linked to the matching id of the subject.

For purposes of the site, the tag search is split into “Filter Artifact Type” and “Filter Person Type” on the frontend.

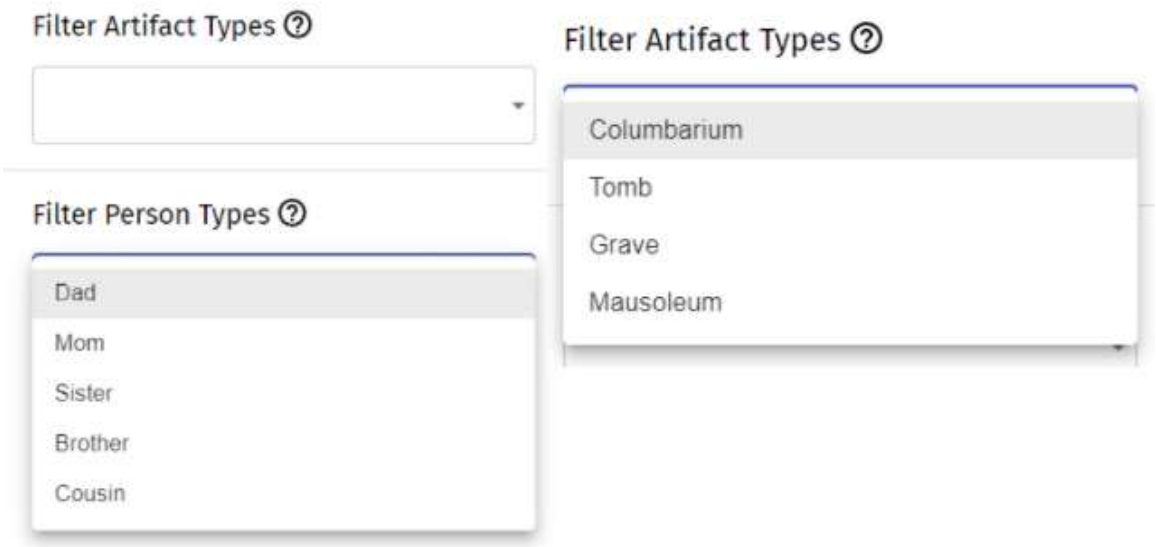


Figure 57: Our Implemented tag search system

# 6.3 Back End API

## 6.3.1 Design Summary

Application programming interfaces or APIs are widely used in software to allow two independent systems to communicate with one another. Using an API adds an extra layer of abstraction and security to an application. With an API, you can hide more complex code away from the user and provide simpler syntax. This results in a more secure application as some users might leave out required information or accidentally break the website. Instead of the website users accessing the database directly, they will contact an API that makes the database

calls for them and verifies that they are valid calls. We will need API endpoints for all functionality that contacts the database [42].

There are a few different architectural styles for APIs. Representational state transfer or REST APIs are popular for web development. We chose a REST API over SOAP or standard communication protocol system due to REST being more secure and using SSL and HTTPS. Our team also has more experience using REST APIs and this is a huge factor due to the short development window and small team.

We will be using Node.js to implement our API. We compared a few different options including Flask and Express and PHP. Flask uses Python and has easy syntax. Flask is a desirable skill to learn for future projects as Python is being more commonly used. The sponsor is currently using Flask for other projects which is why we considered using it for this project. Express is a Node.js framework and is very popular for web development. The team has experience using MERN stacks from Processes of Object Oriented Development, which is why we chose to use it.

Each API call will have a specific method. These are used to perform CRUD (Create, Read, Update, Delete) operations.

**Methods of each request:**

- **GET** - This is the read operation. It will get information from the server.
- **POST** - This is the create operation. It will create a new resource on the server.
- **PUT** - This is the update operation. It must be complete with no empty fields.
- **PATCH** - This is the update operation. It does not need all parameters filled out.
- **DELETE** - This is the delete operation.

For testing and documentation, we will be using Postman. We can make and inspect each API call to check it is working as expected.

Will follow the general HTTP Status codes list [43].

- 200 - Request successful. Use for searches that do not return any results as well.
- 200 - Request received but not completed.
- 204 - Server completed request but no content to return.
- 400 - Bad request.
- 401 - Unauthorized.
- 403 - Administrator user or artifact already exists.
- 404 - Not found.
- 408 - Request timed out.
- 429 - Too many requests in a given time.

### 6.3.2 API Endpoints

<table>
<tr><th colspan="2">Login</th></tr>
<tr><td>Input:<br>{email: string,<br>password: string}</td><td>Errors:<ul><li>Unable to contact database</li><li>Login details do not match an account</li><li>Too many attempts account is locked</li><li>Not email verified</li></ul></td></tr>
<tr><td colspan="2">Description: Login to existing user account. If the account exists and the wrong password is inputted, increment a counter to keep track of unsuccessful login attempts. Will lock the account when 3 incorrect back-to-back login attempts.</td></tr>
<tr><td colspan="2">Returns: Will return success code and login if account info is valid and the account is both email-verified and approved. Returns a failure code otherwise.</td></tr>
</table>

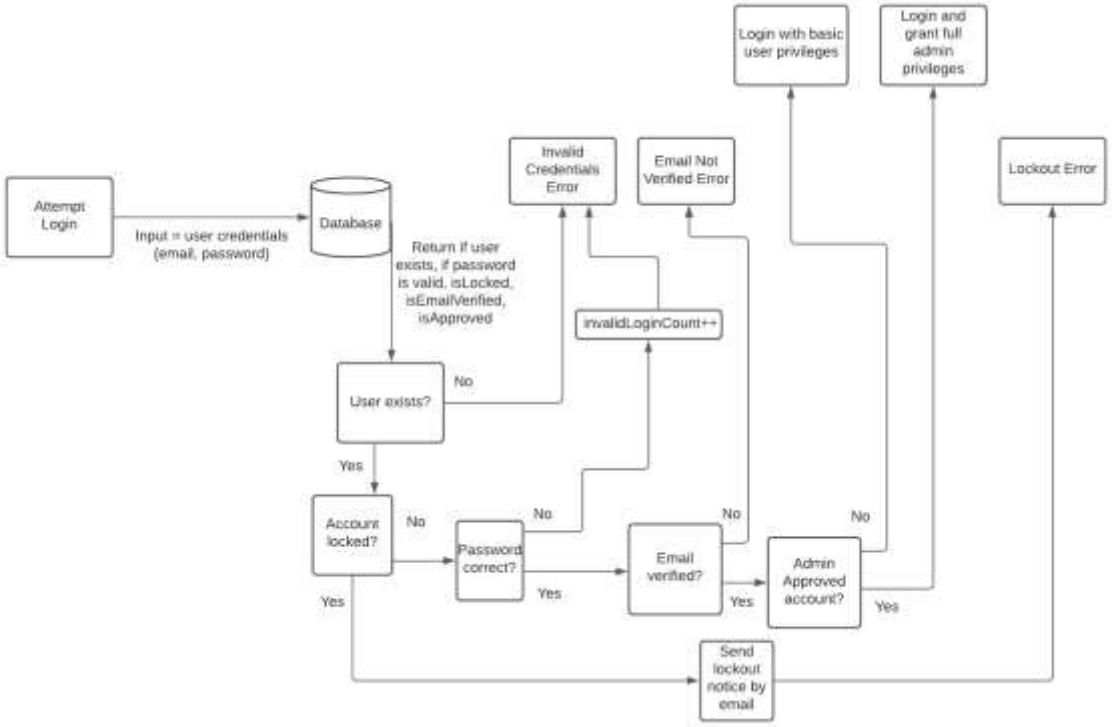


Figure 58: Flowchart Diagram of the Login API

| **Signup** | |
|---|---|
| **Input:**<br>{email: string,<br>password: string} | **Errors:**<br>● Unable to contact database<br>● Email already in use<br>● Password does not match (front end check)<br>● Password too weak (front end check)<br>● All fields not filled in (front end check) |
| **Description:** Signup for a new user account. Will need to verify email before logging in. Once initially created, this will call verify email and send out email verification automatically. | |
| **Returns:** Will return success code if username and email are not taken by other users in the database. Front end will check that the email is valid, all fields are filled out, and password requirements are met. | |

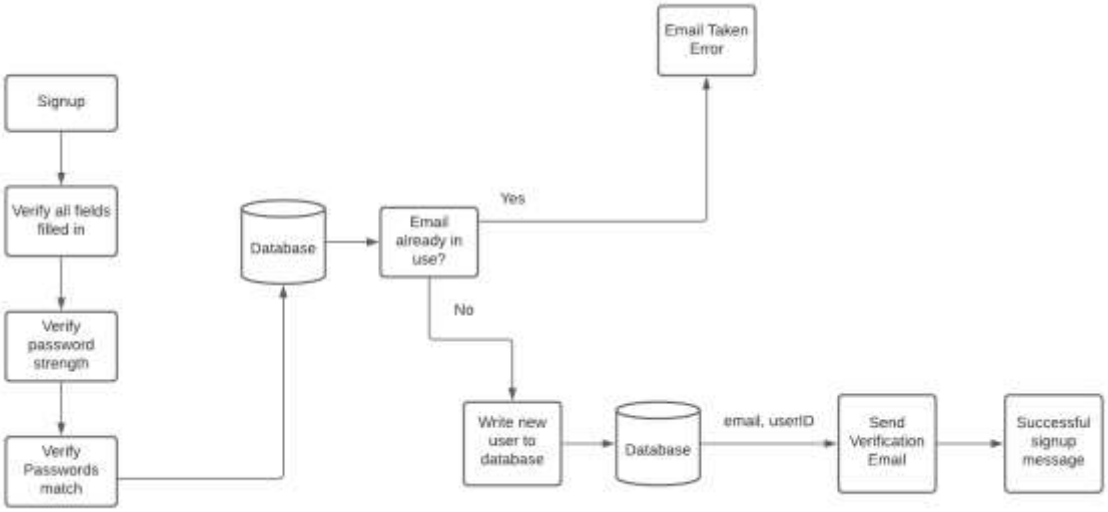


Figure 59: Flowchart Diagram of the Signup API

| **Send Email Verification** | |
|---|---|
| **Input:**<br>{email: string,<br>userID: string} | **Errors:**<br>● Unable to contact database<br>● Email not sent<br>● No user with specified username |
| **Description:** Send the user an email using Twilio's SendGrid transactional email API to verify they have access to email used in signup. Once the link through email is selected, the database will update the Boolean email verified for that specific user. | |
| **Returns:** Success code if username is valid user in the database and email was sent successfully. | |

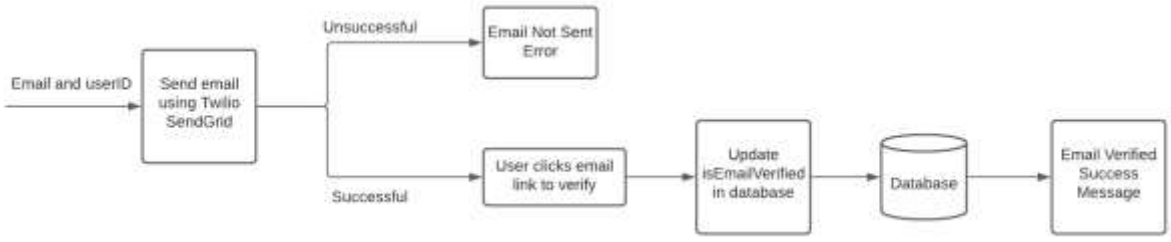


Figure 60: Flowchart Diagram of the Email Verification API

| Send Password Recovery by Email | |
|---|---|
| **Input:**<br>{email: string} | **Errors:**<br>● Unable to contact database<br>● Email not sent<br>● No user with specified email |
| **Description:** On the login page, have an option to select forgot my password. Check if there is a user in the database with a given email. Generic message will be displayed regardless of if the account exists “If the information matches an account with the email address {email} we will send an email to reset your password. If you don't receive an email within a couple of minutes, please check your spam folder.” Where {email} will be what the user inputted. Uses Twilio’s SendGrid transactional email API to send the user an email to update their password. Once the link through email is selected, it will link to a page to create a new password. Same password restrictions apply. Database will be updated to reflect the new password change. | |
| **Returns:** Success code if username is valid user in the database and email was sent successfully. | |

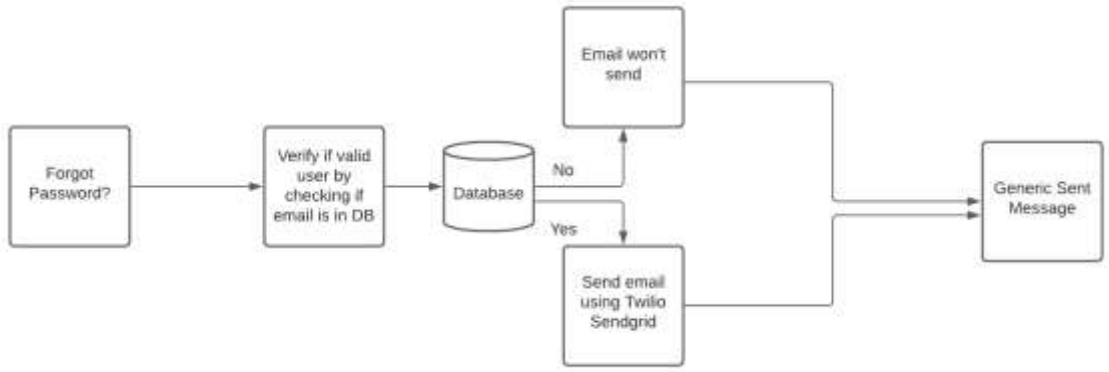


Figure 61: Flowchart Diagram of the Password Recovery API

| Send Lockout Notice by Email | |
|---|---|
| **Input:**<br>{email: string} | **Errors:**<br>● Unable to contact database<br>● Email not sent<br>● No user with specified email |
| **Description:** If too many unsuccessful login attempts, lock the account and send the user an email notifying them their account is locked. The sponsor agreed after 3 unsuccessful, consecutive attempts to lock the account. This will update the user's data isLocked to true. Uses Twilio's SendGrid transactional email API to send email. The link through email will have an option to reset password in order to unlock the account. This will update locked account Boolean and password if changed. | |
| **Returns:** Success code if email is valid and email was sent. | |

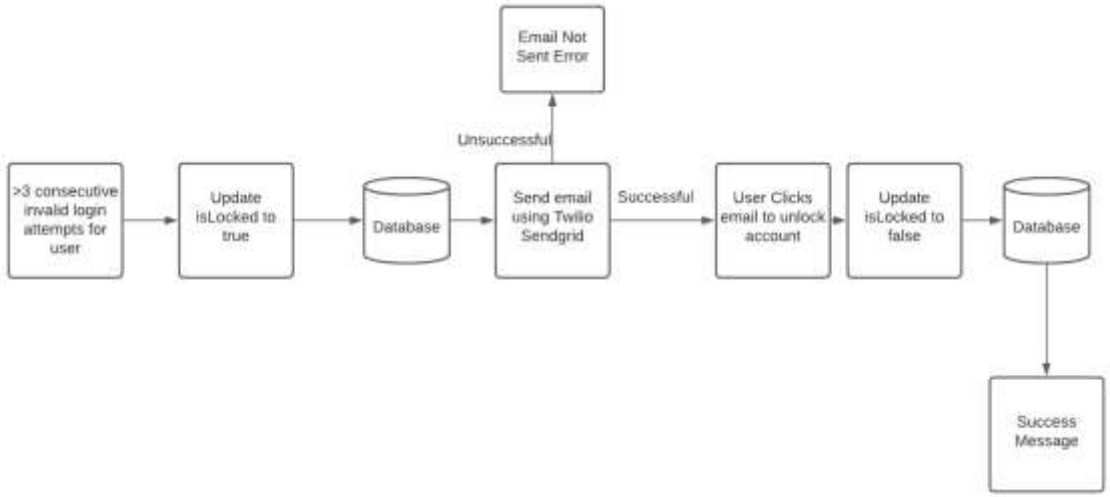


Figure 62: Flowchart Diagram of the Lockout Notice API

| Add Tags (Administrator Only) | |
|---|---|
| **Input:**<br>{newtag: string} | **Errors:**<br>● Tag already exists<br>● Unable to contact database |
| **Description:** Will create a new tag in the database to categorize artifacts, images, people, or places. | |
| **Returns:** Success code if tag is valid. | |

| Edit Tags (Administrator Only) | |
|---|---|
| **Input:**<br>{currentagname: string, newtagname: string} | **Errors:**<br>● Unable to contact database<br>● New tag name already exists |
| **Description:** Will edit the name of a tag already in the database. | |
| **Returns:** Success code if tag rename is valid. | |

| Delete Tags (Administrator Only) | |
|---|---|
| **Input:**<br>{tagname: string} | **Errors:**<br>● Unable to contact database<br>● Tag does not exist |
| **Description:** Will remove an existing tag from the database. | |
| **Returns:** Success code if tag is deleted. | |

| Retrieve User Accounts (Administrator Only) | |
|---|---|
| **Input:**<br>{isadmin: string} | **Errors:**<br>● Unable to contact database<br>● Current user is not an admin |
| **Description:** Get all user accounts in the database for an approved administrator to view. The page will list all user accounts and show the email of the user and if the email is verified or not. It will also show if the account is an admin or basic user. There will be options to promote or demote each user to administrator status after the list is generated. | |
| **Returns:** Will return all queried user accounts for other admins to either promote or demote. Returns success code once list is returned. | |

| Search User Accounts (Administrator Only) | |
|---|---|
| **Input:**<br>{email: string;<br>isadmin: string} | **Errors:**<br>● Unable to contact database<br>● Current user is not an admin |
| **Description:** Search for a specific user in the database. Search will be partial match search. If no results will have a blank page. Once an account is found there will be options to promote or demote each user to administrator status after the list is generated. | |
| **Returns:** Will return all user accounts that have a partial matching email for other admins to either promote or demote. Returns success code once list is returned. | |

| Promote/Demote User Account (Administrator Only) | |
|---|---|
| **Input:**<br>{userID: string,<br>promote: boolean} | **Errors:**<br>● Unable to contact database<br>● User account is already administrator<br>● User account is already basic user |
| **Description:** An administrator account will need to promote other users that sign up to have full functionality on site. They can also demote admins that no longer need admin privileges. Takes userID of selected user, and a Boolean specifying promotion or demotion, and changes isAdmin in the database to 1 (if promoted), or to 0 (if demoted). | |
| **Returns:** Returns success code if successfully able to update isAdmin. | |

| Delete User Account (Administrator Only) | |
|---|---|
| **Input:**<br>{userID: string} | **Errors:**<br>● UserID does not exist |
| **Description:** An administrator account can delete other accounts. Takes userID of selected user from list and deletes the user account from the database. | |
| **Returns:** Returns success code if successfully able to delete the user in the database. | |

| Query to Map Artifacts by Dropping Pinpoints | |
|---|---|
| **Input:**<br>{artifact_title: string,<br>artifact_type: string,<br>submitter_email: string,<br>submitter_name: string,<br>location.country: string,<br>location.state_province: string,<br>location.city: string,<br>location.sitename: string,<br>transcript: string, | **Errors:**<br>● Unable to contact the database<br>● Unable to contact mapping API |

| | |
|---|---|
| translation: string,<br>surname: string,<br>occupation: string<br>given_names: string,<br>person_type:<br>birth_date: string,<br>death_date: string,<br>birthplace.country: string,<br>birthplace.state_province: string,<br>birthplace.city: string,<br>deathplace.country: string,<br>deathplace.state_province: string,<br>deathplace.city: string,<br>cause_of_death: string} | |
| **Description:** When searching, will have an option to map out search results. This will allow the user to search for artifacts and map out the results as pins on a map. The place of birth, place of death, and artifact location in the database will have foreign keys to location tables. The location tables will have fields: country, state / province, city, and sitename. None of the fields are required. Can map one or more specifications and the database will return the longitude and latitude for ALL artifacts matching that search query. If a pin is selected, the site will bring up all information about that artifact. | |
| **Returns:** Will return a list of longitude, latitude and info of all matching results and map out by dropping pins on a map. Success if the database could be contacted. Still successful if an empty result is generated from the search. | |

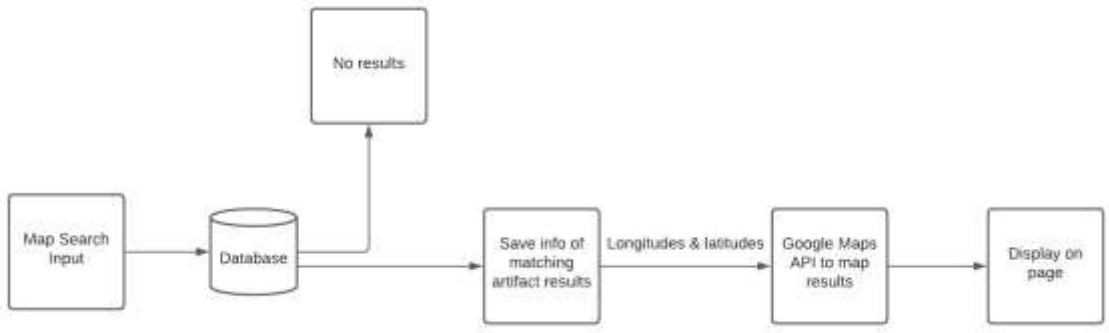


Figure 63: Flowchart Diagram of the Query to Map API

<table>
<tr><th colspan="2">Search for Artifacts</th></tr>
<tr><td>Input:<br>{artifact_title: string,<br>artifact_type: string,<br>submitter_email: string,<br>submitter_name: string,<br>location.country: string,<br>location.state_province: string,<br>location.city: string,<br>location.sitename: string,<br>transcript: string,<br>translation: string,<br>surname: string,<br>given_names: string,<br>person_type: string,<br>occupation: string<br>birth_date: string,<br>death_date: string,<br>birthplace.country: string,<br>birthplace.state_province: string,<br>birthplace.city: string,<br>deathplace.country: string,<br>deathplace.state_province:<br>string,<br>deathplace.city: string,<br>cause_of_death: string}</td><td>Errors:<br>● Unable to contact the database</td></tr>
<tr><td colspan="2">Description: General search for artifacts. one or more specifications and the database will return a page that has ALL artifacts matching that search query. Will have all info for each result, all inputs and images for each artifact listed out in alphabetical order without being mapped out.</td></tr>
<tr><td colspan="2">Returns: Will return a list of all matching search results to list. Success if the database could be contacted. Still successful if an empty result is generated from the search.</td></tr>
</table>

| Delete Artifact (Administrator Only) | |
|---|---|
| **Input:**<br>{artifact_id: string} | **Errors:**<br>● Unable to contact database |
| **Description:** Administrators can delete artifacts after searching and generating results. Will have an X icon to delete an artifact. Will have to select this then confirm with popup. This will delete the artifact and related tables from the database. | |
| **Returns:** Success code is successfully deleted out of the database. | |

| Edit Artifact (Administrator Only) | |
|---|---|
| **Input:**<br>{artifact_id: string,<br>whatever_editing: string} | **Errors:**<br>● Unable to contact database<br>● Required fields not filled out |
| **Description:** Administrators can edit artifacts after searching and generating results. Will have a pen icon to edit an artifact. Change the info you want to update then select done. When editing an artifact, must keep required fields. | |
| **Returns:** Success code is returned when an artifact is edited in the database with required fields submitted. | |

| Upload Artifact (Administrator Only) | |
|---|---|
| **Input:**<br>{artifact_title*: string,<br>arttype*: string,<br>submitter_email: string,<br>submitter_name: string,<br>location.country*: string,<br>location.state_province*: string,<br>location.city*: string,<br>location.sitename*: string,<br>transcript: string,<br>translation: string, | **Errors:**<br>● A required field is empty<br>● Unable to contact database<br>● Unable to contact CHDR server<br>● CHDR server size limit reached<br>● Artifact title already exists in database |

| | |
|---|---|
| images: table,<br>surname: string,<br>given_names: string,<br>person_type: string,<br>occupation: string<br>birth_date: string,<br>death_date: string,<br>birthplace.country: string,<br>birthplace.state_province: string,<br>birthplace.city: string,<br>deathplace.country: string,<br>deathplace.state_province: string,<br>deathplace.city: string,<br>cause_of_death: string} | |
| **Description:** Upload a single artifact as administrator. Must have *REQUIRED fields filled in. We will add images of artifacts to the CHDR server and note the location of where to find them to store in the database. | |
| **Returns:** Success code if uploaded successfully. | |

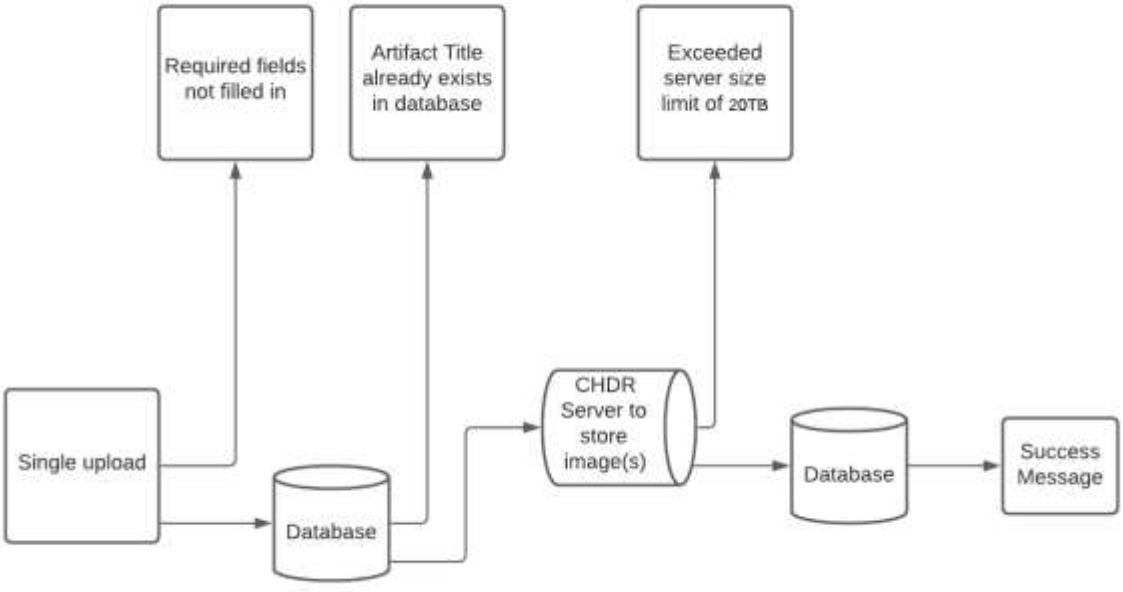


Figure 64: Flowchart Diagram of the Artifact Upload API

| Batch Upload Artifacts (Administrator Only) | |
|---|---|
| **Input:**<br>FOR EACH ARTIFACT<br>{artifact_title*: string,<br>arttype*: string,<br>submitter_email: string,<br>submitter_name: string,<br>location.country*: string,<br>location.state_province*: string,<br>location.city*: string,<br>location.sitename*: string,<br>transcript: string,<br>translation: string,<br>images: table,<br>surname: string,<br>given_names: string,<br>person_type: string,<br>occupation: string<br>birth_date: string,<br>death_date: string,<br>birthplace.country: string,<br>birthplace.state_province: string,<br>birthplace.city: string,<br>deathplace.country: string,<br>deathplace.state_province: string,<br>deathplace.city: string,<br>cause_of_death: string} | **Errors:**<br>● Unable to open / read file<br>● A required field is empty<br>● Unable to contact database<br>● Unable to contact CHDR server<br>● CHDR server size limit reached<br>● Max number of artifacts uploaded at once exceeded<br>● No artifacts in file to upload<br>● Artifact title already exists in database |
| **Description:** Administrators will have the ability to upload artifacts in batches by uploading a document. The admin will be able to upload large amounts of data at once. The maximum number of artifacts uploaded at once will be 5000. Once the file is uploaded, we will show how each field will be added to the database and ask the administrator if this is correct or to manually make changes. This will be an important check in case things are spelled incorrectly on a document or the wrong document was submitted. Once confirmed, we need to check that no | |

| |
|---|
| artifacts with the same title's exist already in the database. Then we will add images of artifacts to the CHDR server and note the location of where to find them to store in the database. |
| **Returns:** Success code if document uploaded successfully. |

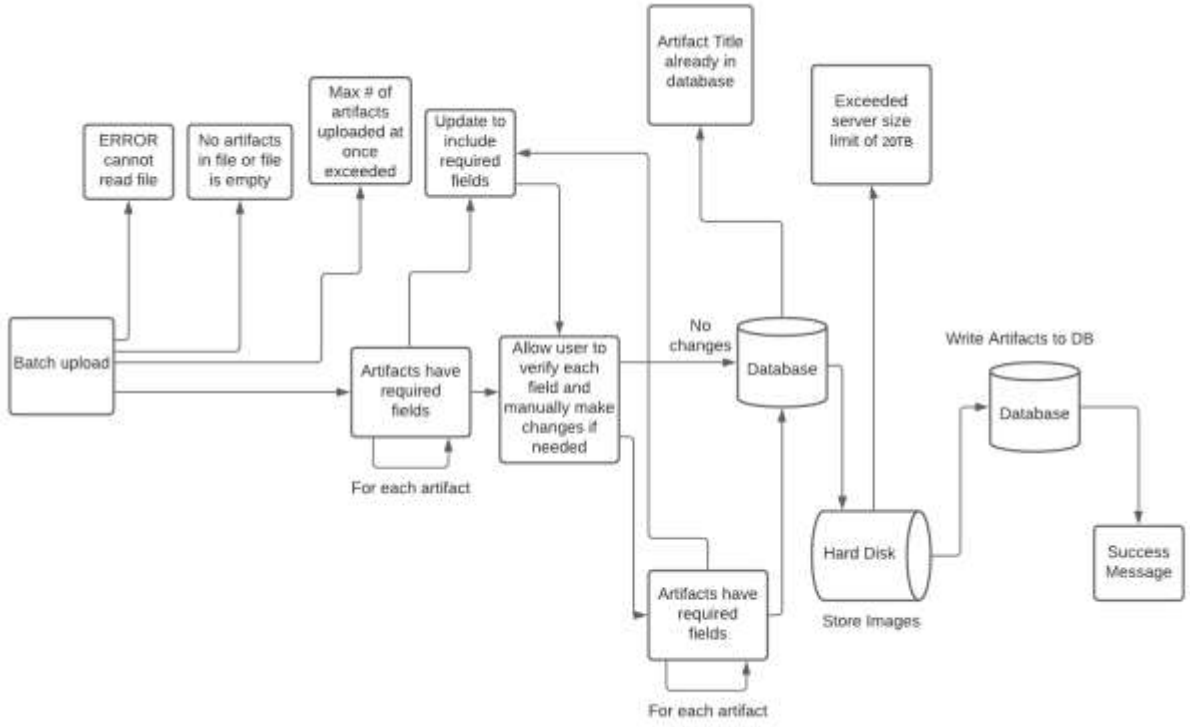


Figure 65: Flowchart Diagram of the Artifact Batch Upload API

### 6.3.2 Final API Endpoints

| post('/api/login') | |
|---|---|
| **email** | Email that the user inputted |
| **password** | The password that the user inputted |

The standard login function. The app checks to see if the account exists with that email and password combination. Once confirmed, the app then checks for the following:

- Check to see if the email has been verified, and if not, then send a message to the user to verify their account.
- Check to see if the account has been locked.

If the account is past those two checks, it is logged in. If the account is locked, then the app sends an email message through sendgrid's api.

| post('/api/signup') | |
|---|---|
| **email** | Email that the user inputted |
| **password** | The password that the user inputted |
| **default_attribution** | The default attribution of the user |

The standard signup function. The app first hashes the password and makes an email token to be used for the verify email function. Then, the app checks to see if there is an account in use with the email provided already. If there is not an account with the given email, then the app uses sendgrid's api to send an email message to the provided email to verify their account, then inputs the user into the database with their info.

| post('/api/password-reset-email') | |
|---|---|
| **email** | Email that the user inputted |

The function to send an email to reset a user's password. If the email is in the database, the app sends an email message, using sendgrid, to the given email address to reset the user's password

| post('/api/password-reset') | |
|---|---|
| **password** | The password that the user inputted |
| **token** | The email token, from the sendgrid email link |

The function to reset a user's password. The app gets the token from the sendgrid reset password email. The app then updates the user's entry in the database to change the password field

| post('/api/get-submitter') | |
|---|---|
| **submitter_id** | The internal id of a submitter in the database, usually taken from an artifact entry. |

Returns a json object of the submitter with the given id.

| post('/api/get-person') | |
|---|---|
| **person_id** | The internal id of a person in the database, usually taken from an artifact entry. |

Returns a json object of the person with the given id.

| post(‘/api/get-people-from-artifact’) | |
|---|---|
| **artifact_id** | The internal id of an artifact in the database, usually taken from its own entry. |

Returns an array of json objects that contain entries of people in the database, where their artifact_link field equals the given artifact id

| post(‘/api/get-location’) | |
|---|---|
| **location_id** | The internal id of a location in the database, usually taken from an artifact entry. |

Returns a json object of the location with the given id.

| post(‘/api/get-collection’) | |
|---|---|
| **collection_id** | The internal id of a collection in the database, usually taken from an image entry. |

Returns a json object of the collection with the given id.

| post(‘/api/get-image’) | |
|---|---|
| **image_id** | The internal id of an image in the database, usually taken from an artifact entry. |

Returns a json object of the image with the given id.

| post(‘/api/get-images-from-artifact’) | |
|---|---|
| **artifact_id** | The internal id of an artifact in the database, usually taken from its own entry. |

Returns an array of json objects that contain entries of image in the database, where their artifact_link field equals the given artifact id.

| post('/api/get-attribution-from-image') | |
|---|---|
| **attribution_link** | The internal id of an attribution in the database, usually taken from an image entry. |

Returns a json object of the attribution with the given id.

| post('/api/get-artifact-tags') | |
|---|---|
| **art_id** | The internal id of an artifact in the database, usually taken from its own entry. |

Returns an array of json objects that contain entries of tags in the database, where they are linked to the artifact with the given id.

| post('/api/get-person-tags') | |
|---|---|
| **ppl_id** | The internal id of a person in the database, usually taken from its own entry. |

Returns an array of json objects that contain entries of tags in the database, where they are linked to the person with the given id.

| post('/api/advanced-search') | |
|---|---|
| **page** | The page number of the artifacts list, used to paginate the list for front end. |
| **artifact_title** | The name of the artifact |
| **submitter_id** | The id of a submitter |

| | |
|---|---|
| **first_name** | First name of a given person |
| **last_name** | Last name of a given person |
| **latitude** | General latitude, usually geo-coded through an app like leaflet REQUIRED if longitude is not empty or null |
| **longitude** | General longitude, usually geo-coded through an app like leaflet REQUIRED if latitude is not empty or null |
| **distance** | The maximum amount of distance away from the latitude/longitude point. REQUIRED if latitude and longitude are not empty or null |
| **distance_unit** | Two entries: “miles” and “kilometers”. Used for distance calculations REQUIRED if distance is not empty or null |
| **include_birth_location** | If set to “true”, then the search is based on a person’s location of birth |
| **include_death_location** | If set to “true”, then the search is based on a person’s location of death |
| **include_artifact_location** | If set to “true”, then the search is based on the location of an artifact |
| **country** | Name of a country |
| **state_province** | Name of a state or province |
| **city** | Name of a city |
| **site_name** | Name of a site that is hosting a given artifact |

| | |
|---|---|
| **artifact_type** | A name, or array of names, of tags related to an artifact |
| **person_type** | A name, or array of names, of tags related to a person |
| **birth_date** | A given date of birth |
| **birth_timeframe** | Four possible entries: “Approximate” searches between the previous year and next year of the given birth date, “Exact” searches for the exact date, “Before” searches for any date before the given date, and “After” searches for any date after the given date. REQUIRED if birth_date is not empty or null. |
| **death_date** | A given date of death |
| **death_timeframe** | Four possible entries: “Approximate” searches between the previous year and next year of the given death date, “Exact” searches for the exact date, “Before” searches for any date before the given date, and “After” searches for any date after the given date. REQUIRED if death_date is not empty or null. |
| **sort_by** | Three entries: “Relevancy” currently sorts by placement in the database, “Artifact Title (A-Z)” sorts the artifacts in alphabetical order, “Artifact Title (Z-A)” sorts the artifacts in reverse alphabetical order. If empty or null, the app sorts by Relevancy by default. |

Given the parameters above, the app searches the database for all entries that fit all of the filters, with the hierarchy of filters being in the order they appear above, with the exception of the page parameter. Most of the filters just compare the parameters given with entries in a database, exceptions are as follows: For a geolocational search, the geolocational data is usually converted by the front end to be used in the back end. Then, the distance being passed into the app is converted from the given distance unit to the geolocational distance. For a dating search, the app filters based on either the exact date of birth/death, before, or after.

After all of the filters, the app would be able to filter between alphabetical or reverse alphabetical order. After the sorting, the app would then cut the results to 20, with an offset being 20 multiplied by the page parameter.

| **post('/api/search-by-artifact-id')** | |
|---|---|
| **artifact_id** | The internal id of an artifact in the database, usually taken from its own entry. |

Returns all of the information for an artifact if the artifact_id entered returns a match otherwise informs the frontend the artifact doesn't exist.

| **post ('/api/add-single-artifact')** | |
|---|---|
| **title*** | The artifacts title. |
| **transcript** | The artifacts transcript. |
| **translation** | The artifacts translation. |
| **artifact_type*** | A 2D array of artifact types where the array corresponds to the artifact being added. |
| **artifact_country** | The country where the artifact is located. |
| **artifact_state_province** | The state the artifact is located in. |
| **artifact_city** | The city where the artifact is located. |
| **artifact_sitename** | The site where the artifact is located. |
| **artifact_latitude** | The latitude the artifact is located. |
| **artifact_longitude** | The longitude of the artifact is located. |

| | |
|---|---|
| **person_type** | A 2D of person types where each array corresponds to each person being added. |
| **surname*** | An array of surnames that correspond to each person tied to the artifact. |
| **given_name** | An array of given names that correspond to each person tied to the artifact. |
| **occupation** | An array of occupations that correspond to each person tied to the artifact. |
| **birth_date** | An array of birth dates that correspond to each person tied to the artifact. |
| **death_date** | An array of death dates that correspond to each person tied to the artifact. |
| **cause_of_death** | An array of the cause of death that corresponds to each person tied to the artifact. |
| **birth_country** | An array of birth countries that correspond to each person tied to the artifact. |
| **birth_state_province** | An array of birth states that correspond to each person tied to the artifact. |
| **birth_city** | An array of birth cities that correspond to each person tied to the artifact. |
| **birth_sitename** | An array of birth sites that correspond to each person tied to the artifact. |
| **birth_latitude** | An array of birth latitudes that correspond to each person tied to the artifact. |
| **birth_longitude** | An array of birth longitudes that correspond to each person tied to the artifact. |

| | |
|---|---|
| **death_country** | An array of death countries that correspond to each person tied to the artifact. |
| **death_state_province** | An array of death states that correspond to each person tied to the artifact. |
| **death_city** | An array of death cities that correspond to each person tied to the artifact. |
| **death_sitename** | An array of death sites that correspond to each person tied to the artifact. |
| **death_latitude** | An array of death latitudes that correspond to each person tied to the artifact. |
| **death_longitude** | An array of death longitudes that correspond to each person tied to the artifact. |
| **submitter_name*** | The name of the person who is submitting the artifact. |
| **email*** | The email of the person who is submitting the artifact. |
| **isTwitter*** | A boolean for whether the artifact submitted was entered through twitter. |
| **description** | An array of descriptions corresponding to each image being uploaded with the artifact. |
| **attribution** | An array of attributions corresponding to each image being uploaded with the artifact. |
| **image_country** | An array of image countries that correspond to each image tied to the artifact. |
| **image_state_province** | An array of image states that correspond to each image tied to the artifact. |
| **image_city** | An array of image cities that correspond to each image tied to the artifact. |

| | |
|---|---|
| **image_sitename** | An array of image sites that correspond to each image tied to the artifact. |
| **image_latitude** | An array of image latitudes that correspond to each image tied to the artifact. |
| **image_longitude** | An array of image latitudes that correspond to each image tied to the artifact. |

The API starts by checking to see if the user is an admin and then if the artifact title has already been taken. Afterwards it goes through the uploaded information and adds it all to the database. Returns a statement that an “Artifact has been added” if there are no errors, it will return “User is not an admin” if the user is not an admin, “Artifact title already exists in database” if the artifact title already exists, and otherwise it will return an error. All the parameters are required to be entered have an * in the table and there must be an image included in the upload.

| **post (‘/api/delete-artifact’)** | |
|---|---|
| **title** | The artifacts title. |

The API starts by checking to see if the user is an admin and will return “User is not an admin” if the user is not an admin. Afterwards it will check to see if the artifact is a part of a collection, then it will go through and delete all the information of the artifact that isn’t associated with another artifact. The order of information being deleted was images, image locations, person tags, people, birth locations, death locations, artifact tags, artifact table, collection (if the artifact is a part of and the last artifact in a collection), submitter table, and artifact location.

| **post (‘/api/delete-users’)** | |
|---|---|
| **user_id** | The id associated with the user to be deleted. |

The function to delete a user. Searches a user_id in the database, if it is found then the user is deleted and the statement "User has been deleted." is returned. If

the user is not an admin it returns “User is not an admin” and if they are not logged in it returns “Token not valid”.

| post (‘/api/edit-artifact) | |
|---|---|
| **title*** | The artifacts title. |
| **transcript** | The artifacts transcript. |
| **translation** | The artifacts translation. |
| **artifact_type** | A 2D array of artifact types where the array corresponds to the artifact being added. |
| **artifact_country** | The country where the artifact is located. |
| **artifact_state_province** | The state the artifact is located in. |
| **artifact_city** | The city where the artifact is located. |
| **artifact_sitename** | The site where the artifact is located. |
| **artifact_latitude** | The latitude the artifact is located. |
| **artifact_longitude** | The longitude of the artifact is located. |
| **person_type** | A 2D of person types where each array corresponds to each person being added. |
| **surname** | An array of surnames that correspond to each person tied to the artifact. |
| **given_name** | An array of given names that correspond to each person tied to the artifact. |
| **occupation** | An array of occupations that correspond to each person tied to the artifact. |

| | |
|---|---|
| **birth_date** | An array of birth dates that correspond to each person tied to the artifact. |
| **death_date** | An array of death dates that correspond to each person tied to the artifact. |
| **cause_of_death** | An array of the cause of death that correspond to each person tied to the artifact. |
| **birth_country** | An array of birth countries that correspond to each person tied to the artifact. |
| **birth_state_province** | An array of birth states that correspond to each person tied to the artifact. |
| **birth_city** | An array of birth cities that correspond to each person tied to the artifact. |
| **birth_sitename** | An array of birth sites that correspond to each person tied to the artifact. |
| **birth_latitude** | An array of birth latitudes that correspond to each person tied to the artifact. |
| **birth_longitude** | An array of birth longitudes that correspond to each person tied to the artifact. |
| **death_country** | An array of death countries that correspond to each person tied to the artifact. |
| **death_state_province** | An array of death states that correspond to each person tied to the artifact. |
| **death_city** | An array of death cities that correspond to each person tied to the artifact. |
| **death_sitename** | An array of death sites that correspond to each person tied to the artifact. |

| | |
|---|---|
| **death_latitude** | An array of death latitudes that correspond to each person tied to the artifact. |
| **death_longitude** | An array of death longitudes that correspond to each person tied to the artifact. |
| **submitter_name** | The name of the person who is submitting the artifact. |
| **email** | The email of the person who is submitting the artifact. |
| **isTwitter** | A boolean for whether the artifact submitted was entered through twitter. |
| **description** | An array of descriptions corresponding to each image being uploaded with the artifact. |
| **attribution** | An array of attributions corresponding to each image being uploaded with the artifact. |
| **image_country** | An array of image countries that correspond to each image tied to the artifact. |
| **image_state_province** | An array of image states that correspond to each image tied to the artifact. |
| **image_city** | An array of image cities that correspond to each image tied to the artifact. |
| **image_sitename** | An array of image sites that correspond to each image tied to the artifact. |
| **image_latitude** | An array of image latitudes that correspond to each image tied to the artifact. |
| **image_longitude** | An array of image latitudes that correspond to each image tied to the artifact. |
| **artifact_id** | The internal id of an artifact in the database |

| | |
|---|---|
| **person_id** | The internal id of a person who is associated with the artifact, if the array has a new value it is a new person |
| **image_id** | The internal id of an image who is associated with the artifact, if the array has a new value it is a new image |

The function edits an artifact. The function starts by updating the artifact title, transcript, translation, the artifact's location and the submitter table. Then the artifact tags that are inputted are compared to the database, any deleted are to be deleted, any to be updated will update and then any new ones will be added.
With more than one person the person_ids that are inputted are compared to the database, any that don't exist are to be deleted, any to be updated will update and then any new ones will be added as the person is being added, deleted, or updated the person tags will also be added deleted or updated. If there is only one person to be updated then the person will be updated and afterwards its tags will be added, deleted, or updated. Any changes to a person's location will occur before the person itself is.

Then loop through the image_ids and find the differences between what is inputted and what is in the database, any that don't exist are to be deleted, any to be updated will update and then any new ones will be added. If they are removed, they will also be removed from the server. Any changes to an image location will occur before the image itself is.

If the user is not an admin it returns "User is not an admin" and if they are not logged in it returns "Token not valid".

| **post ('/api/search-users')** | |
|---|---|
| **search** | The string that is going to be compared to all the user table's emails. |
| **page** | The number page that is going to be used for pagination. |

Returns an array of json objects that contain entries of users in the database based on the inputted parameter search compared to the email value in the database. Then page value will be used for pagination so only twenty or less artifacts will be returned. If the user is not an admin it returns “User is not an admin” and if they are not logged in it returns “Token not valid”.

| **post (‘/api/promote-demote’)** | |
|---|---|
| **email** | The email of the user being edited. |
| **admin_status** | A boolean value for if the user is an admin or not. |

Returns status code 200 if successful. The API takes the email and updates the admin status of the user associated with the email. If the user is not an admin it returns “User is not an admin” and if they are not logged in it returns “Token not valid”.

| **post (‘/api/batch-upload’)** | |
|---|---|
| **title** | The artifacts title. |
| **transcript** | The artifacts transcript. |
| **translation** | The artifacts translation. |
| **artifact_type** | A 2D array of artifact types where the array corresponds to the artifact being added. |
| **artifact_country** | The country where the artifact is located. |
| **artifact_state_province** | The state the artifact is located in. |
| **artifact_city** | The city where the artifact is located. |
| **artifact_sitename** | The site where the artifact is located. |

| | |
|---|---|
| **artifact_latitude** | The latitude the artifact is located. |
| **artifact_longitude** | The longitude of the artifact is located. |
| **person_type** | A 2D of person types where each array corresponds to each person being added. |
| **surname** | An array of surnames that correspond to each person tied to the artifact. |
| **given_name** | An array of given names that correspond to each person tied to the artifact. |
| **occupation** | An array of occupations that correspond to each person tied to the artifact. |
| **birth_date** | An array of birth dates that correspond to each person tied to the artifact. |
| **death_date** | An array of death dates that correspond to each person tied to the artifact. |
| **cause_of_death** | An array of the cause of death that corresponds to each person tied to the artifact. |
| **birth_country** | An array of birth countries that correspond to each person tied to the artifact. |
| **birth_state_province** | An array of birth states that correspond to each person tied to the artifact. |
| **birth_city** | An array of birth cities that correspond to each person tied to the artifact. |
| **birth_sitename** | An array of birth sites that correspond to each person tied to the artifact. |
| **birth_latitude** | An array of birth latitudes that correspond to each person tied to the artifact. |

| | |
|---|---|
| **birth_longitude** | An array of birth longitudes that correspond to each person tied to the artifact. |
| **death_country** | An array of death countries that correspond to each person tied to the artifact. |
| **death_state_province** | An array of death states that correspond to each person tied to the artifact. |
| **death_city** | An array of death cities that correspond to each person tied to the artifact. |
| **death_sitename** | An array of death sites that correspond to each person tied to the artifact. |
| **death_latitude** | An array of death latitudes that correspond to each person tied to the artifact. |
| **death_longitude** | An array of death longitudes that correspond to each person tied to the artifact. |
| **submitter_name** | The name of the person who is submitting the artifact. |
| **email** | The email of the person who is submitting the artifact. |
| **isTwitter** | A boolean for whether the artifact submitted was entered through twitter. |
| **description** | An array of descriptions corresponding to each image being uploaded with the artifact. |
| **attribution** | An array of attributions corresponding to each image being uploaded with the artifact. |
| **image_country** | An array of image countries that correspond to each image tied to the artifact. |
| **image_state_province** | An array of image states that correspond to each image tied to the artifact. |

| | |
|---|---|
| **image_city** | An array of image cities that correspond to each image tied to the artifact. |
| **image_sitename** | An array of image sites that correspond to each image tied to the artifact. |
| **image_latitude** | An array of image latitudes that correspond to each image tied to the artifact. |
| **image_longitude** | An array of image latitudes that correspond to each image tied to the artifact. |
| **number** | The number artifact in the CSV file the current line of the CSV file is associated with, |
| **filename** | The name of the image file. |

The API takes a zip file with all of the parameters listed above except for submitter_name and email which are submitted on the website individually. The zip file and the images will all be saved in the same place on the backend. The csv file is then unzipped, read, and at the end of the function the csv file will be deleted. Any error will delete the csv file and the collection of images. If there is an error in formatting “Improper Formatting at line #" will be returned where # will be the line of the CSV file where there is an error.

The CSV file is read, and each row will be stored in its corresponding parameter. Checks are done to ensure that all parameters that must be filled are so that there are no errors during upload. The collection will be inserted first followed by the submitter. Then we loop through all the artifacts inserting the artifact table information and the artifact’s location. In the loop there will be a loop for the current row of the CSV file and every row afterwards until the artifact title value is not empty or null.

Inside this loop it will enter all the information associated with people after checking to see if the person type and surname is not null. The loop then checks to see if there is an image associated and will insert the image if it is. If the user is not an admin it returns “User is not an admin” and if they are not logged in it returns “Token not valid”.

| post(‘/api/search-tag’) | |
|---|---|
| **submitter_id** | The internal id of a submitter in the database, usually taken from an artifact entry. |
| **type** | The type of tag whether a person or an artifact |
| **Page** | The page number for pagination. |

Searches the tag table for tag and type of tag Returns an array of 20 or less json objects that are returned from the search. If user is not logged in then “Token not valid” is returned.

| post(‘/api/add-tag’) | |
|---|---|
| **tag_name** | The name of the tag to be submitted |
| **tag_type** | The type of tag whether a person or an artifact |

Returns a status code of 200 if the tag is successfully added to the database or if the tag is already in the database. If the user is not logged in then “Invalid login” is returned.

| post(‘/api/delete-tag’) | |
|---|---|
| **tag_id** | The unique id of the tag to be deleted. |

Returns a status code of 200 if the tag is successfully deleted from both the tag table and the tag2artifact or tag2person table. If user is not logged in then “Invalid login” is returned.

| post(‘/api/edit-tag’) |
|---|

| | |
|---|---|
| **tag_id** | The unique id of the tag to be deleted. |
| **new_name** | The new name for the tag of the given id. |

Returns a status code of 200 if the tag is successfully edited in both the tag table and the tag2artifact or tag2person table depending on what the tag_id is associated with If user is not logged in then “Invalid login” is returned. “Invalid tag” is returned if the tag_id is not in the database.

# 7 Testing Details

## 7.1 Unit Testing

For testing we are implementing the Jest testing framework. Jest is going to be installed with “npm install --save-dev jest” and the following line will be added to package.json: "scripts": { "test": "jest" } . Now the way Jest works it expects the word test to be in the name of the file so if we have a file named sum.js then we can name the test file sum.test.js. The Figures 66 and 67 are an example of a simple addition function and its associated test [44].

```
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Figures 66 (top) and 67 (bottom): A Sample Function and Associated Test

The test function inputs parameters into the actual function and compares the result of the function to the value in the “.tobe()” field. When the test is run using “npm run test” it will return whether each test passed or failed an example of a passed test is shown below in Figure 68.

```
PASS  ./sum.test.js
✓ adds 1 + 2 to equal 3 (5ms)
```

Figure 68: Example Output of a Passed Unit Test

If the test fails Jest will tell us what the expected value was and what the value it received was which will help us find any bugs. There are also many types of matchers that we will need to use for the website. The matcher .tobe is used to test the equality of string and numbers while an array needs to be tested using the checker toEqual. There is also toBeFalsy() which will match with anything an if statement treats as false such as 0, NULL, or undefined and toBeTruthy() which accepts anything that an if statement does not consider false. Jest can also be used to test async functions, but the code is written up slightly differently for these because jest expects a synchronous function unless it is told otherwise. For testing a callback function we must use the argument done(). Jest will not finish the test until the done() argument is called. If done() is never called, then there will be a timeout error and the test will fail. One issue with this though is when the failure comes from the expect statement. When the expect statement fails and done() is never called Jest will output that the error was a timeout error even though that is not the case. So when testing for a callback we will be using a try block. The try block will go around the expect argument so if there is an error it will be passed to the catch block and into done() so the error can be read. Figure 69 is an example of a properly formatted callback Unit Test [45].

```
test('the data is peanut butter', done => {
  function callback(data) {
    try {
      expect(data).toBe('peanut butter');
      done();
    } catch (error) {
      done(error);
    }
  }

  fetchData(callback);
});
```

Figure 69: A Unit Test for a Callback Function

Promises are much easier to test for than callbacks. If a promise is returned from a function to a test Jest will wait until the promise is resolved to evaluate the test. The argument then() is also used here to evaluate what is expected from the

returned promise. Figure 70 shows an example of how Jest tests for promises using then().

```
test('the data is peanut butter', () => {
  return fetchData().then(data => {
    expect(data).toBe('peanut butter');
  });
});
```

Figure 70: A Unit Test for a Promise Using then()

Instead of using then(), resolves and rejects can also be used for evaluating a promise. With the .resolves matcher inside of the expect statement Jest will not finish the test until the promise is resolved. If the promise is never resolved or the promise is rejected the test will fail. Opposite to this is the .rejects matcher which will fail whenever a promise is accepted. This can be very useful when testing for errors as shown in Figure 71, which is a test function that passes if the promise is rejected and fails if the promise containing an error is returned.

```
test('the fetch fails with an error', () => {
  return expect(fetchData()).rejects.toMatch('error');
});
```

Figure 71: A Unit Test for a Promise Using .rejects

The last way to test an asynchronous function is by using the arguments async or await. We declare the function as a test function at the beginning using async and then use the await argument right before the expect or fetchData argument. Async and Await can also be used in combination with .resolve and .rejects and shown in Figure 72.

```
test('the data is peanut butter', async () => {
  await expect(fetchData()).resolves.toBe('peanut butter');
});

test('the fetch fails with an error', async () => {
  await expect(fetchData()).rejects.toMatch('error');
});
```

Figure 72: A Unit Test Bombining Async/Await with .resolves & .rejects

## 7.2 System Testing

For system testing we are going to be using the tool Selenium or more specifically Selenium Webdriver. The tool is open source meaning it will be free to use for our project. There are four sections which are Selenium's client library, JSON's wire protocol over HTTP, Browser Drivers, and Browsers [46]. The client library is what allows Selenium to work with different languages or in our case JavaScript. JSON wire protocol is what Selenium uses to transfer data from the client to the web server. The browser drivers are what reads the Selenium commands and runs the commands inside of the browser and then the browsers are the internet browsers we all use. The setup for Selenium is simple as all that is needed is node.js and an IDE. To install Selenium Webdriver you need to enter this command into the terminal: npm install --save selenium-webdriver chromedriver geckodriver. This specific command includes the browser drivers for Google Chrome and Firefox. The dependencies will automatically be installed into package.json. Now Figure 73 shows one way Selenium Webdriver can be configured to do a UI test [47]. Whenever we use Selenium, we will need to use the Builder to create a new browser. In the example below google is loaded up and the query is sent and run through google. THen the query string is grabbed and compared against the expected query. If there is a match, then the console will print out "Match Status True " and lastly the browser will be exited. This use of Selenium Webdriver can be useful to our project as users will be running search queries when searching for artifacts in our database and search queries will be running when loading a map of artifacts on Leaflet.

```
const {By,Key,util, Builder} = require("selenium-webdriver");
require("chromedriver");
 async function example(){
 // The string variable containing value to search
 var searchString = "Javascript strings with Selenium webdriver";
     //To wait for browser to build and launch properly
 let driver = await new Builder().forBrowser("chrome").build();
    //To fetch http://google.com from the browser with our code.
 await driver.get("http://google.com");
   //To send a search query by passing the value in searchString.
 await driver.findElement(By.name("q")).sendKeys(searchString,Key.RETURN);
     //To fetch the value of currentUrl and storing it into a variable by converting it to string
 var url= (await driver.getCurrentUrl()).toString();
   //To find the start and position of query string
 var start = url.indexOf("?q=");
 var end = url.indexOf("&");
 //To extract the query string wrt the start and end positions
 var queryString = url.slice(start+3,end);
 //To get an array containing all keywords by splitting with '+'
 queryStringArray = queryString.split("+");
 console.log("ACTUAL-",queryStringArray);
 //To get an array containing words of original searchString
 expectedArray=searchString.split(" ");
 console.log("EXPECTED-",expectedArray);
 //To compare the expectedArray with the Actual query string array
 console.log("Match Status:",JSON.stringify(queryStringArray)==JSON.stringify(expectedArray));
    //It is always a safe practice to quit the browser after execution
 await driver.quit();
}
 example();
```

Figure 73: Using Selenium Webdriver to test UI through a Query String

We can also use Selenium to add elements to our UI and quickly see how they look. Instead of running By.name(“q”) on our website we can run a command such as By.id(“sampletext”). This can be used to load a browser with a new test in a field that has that id, it does not even need to be tested against anything, but is just a way to quickly look at the UI. Selenium Webdriver can be used as a way to perform End-to-End testing in a similar way. For example, in Figure 74, the code written runs a test that will load the login page and then enter the username and password into the respective boxes [48]. After this the login button will be clicked logging the user in and taking them to a new page where the function of this page is printed to the console.

```
var webdriver = require('selenium-webdriver');
var By = webdriver.By;
var until = webdriver.until;

var driver = new webdriver.Builder().forBrowser('chrome').build();
driver.manage().window().maximize();

driver.get('https://forio.com/epicenter/sign-in');

var loginId = driver.findElement({xpath:'//input[@name="email"]'});
        loginId.sendKeys('harishyadav.lavrick@gmail.com');

var passwd = driver.findElement(By.xpath("//input[@name='password']"));

        passwd.sendKeys('Salman@15');

var submit = driver.findElement(By.xpath("//div[@id='register-or-sign-in']//button"));
        submit.click();

driver.getTitle().then(function(title) {console.log(title);});

driver.quit();
```

Figure 74: Selenium System Test for Login

This can work for our tests by allowing us to test for login or any other features that require inputs such as searching or making sure links are working. Rather than printing function titles to console we can test if the expected URL matches the url we are at using an Assert. Along with this we are going to use Selenium IDE to record some tests. Selenium IDE works as an extension in Google Chrome and Firefox for running test cases. This works best once a section of the website is already running correctly. This type of test is also much simpler to run because it requires no code, and the tests we create will be given to the sponsor at the end of the project to assist them in bug finding if an error occurs in the website after they make a change or if something breaks down the line. The sponsors will have the ability to easily create their own tests using this tool if they desire to. To use Selenium IDE go to the website https://www.selenium.dev/selenium-ide/ and install one of the extensions [49]. From there we create a new project and enter in a base URL which for us is going to be the gravestone project website. After this is entered, click record to start a new test. The base url will be loaded onto a new page and in the bottom

right-hand corner it will say “Selenium IDE is recording” as shown in Figure 75 [5]:

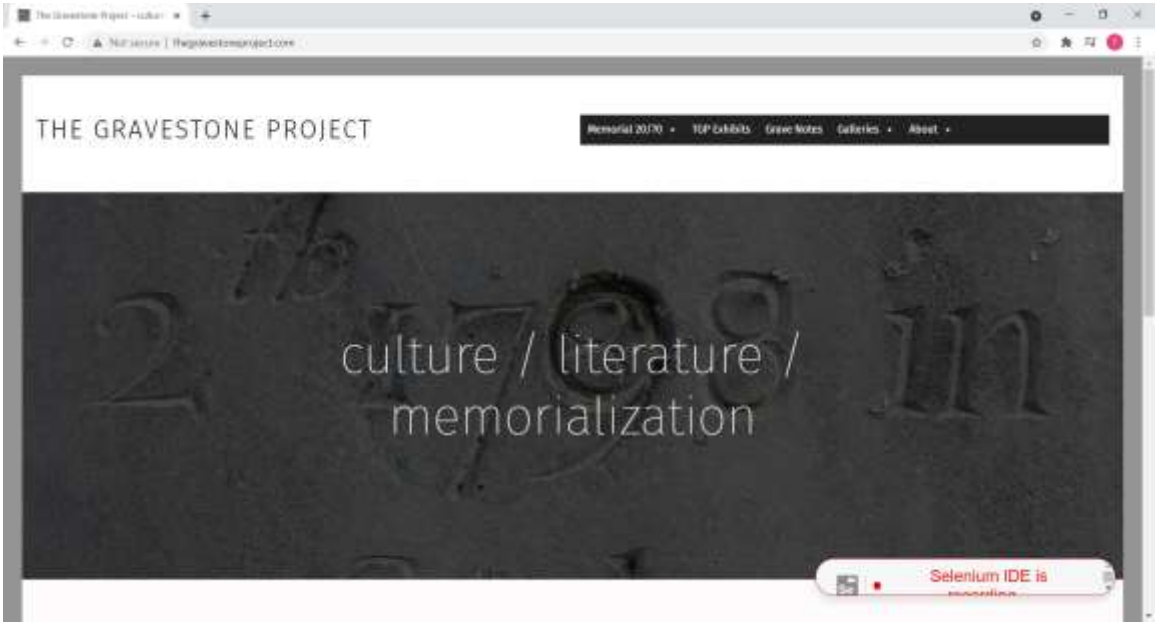


Figure 75: Selenium IDE Recording

To stop the recording click back on the Selenium IDE and click stop recording. Then name the test and the test is saved to run as many times as you like. In our case we can rerun the test whenever we update a piece of code to guarantee that it is still working. Selenium IDE loads the website when a test is run and goes through every input that was made so we can watch the test as it happens and see exactly where an error might have occurred. If a test is completed correctly every command run will be listed in the Log with a green “Ok” written next to it. If the test fails, the command will be shown in red and an error message outlining what went wrong will be shown in the Log section. This is shown in Figure 76 where there is a timeout error on the gravestone project when looking for the origins button.

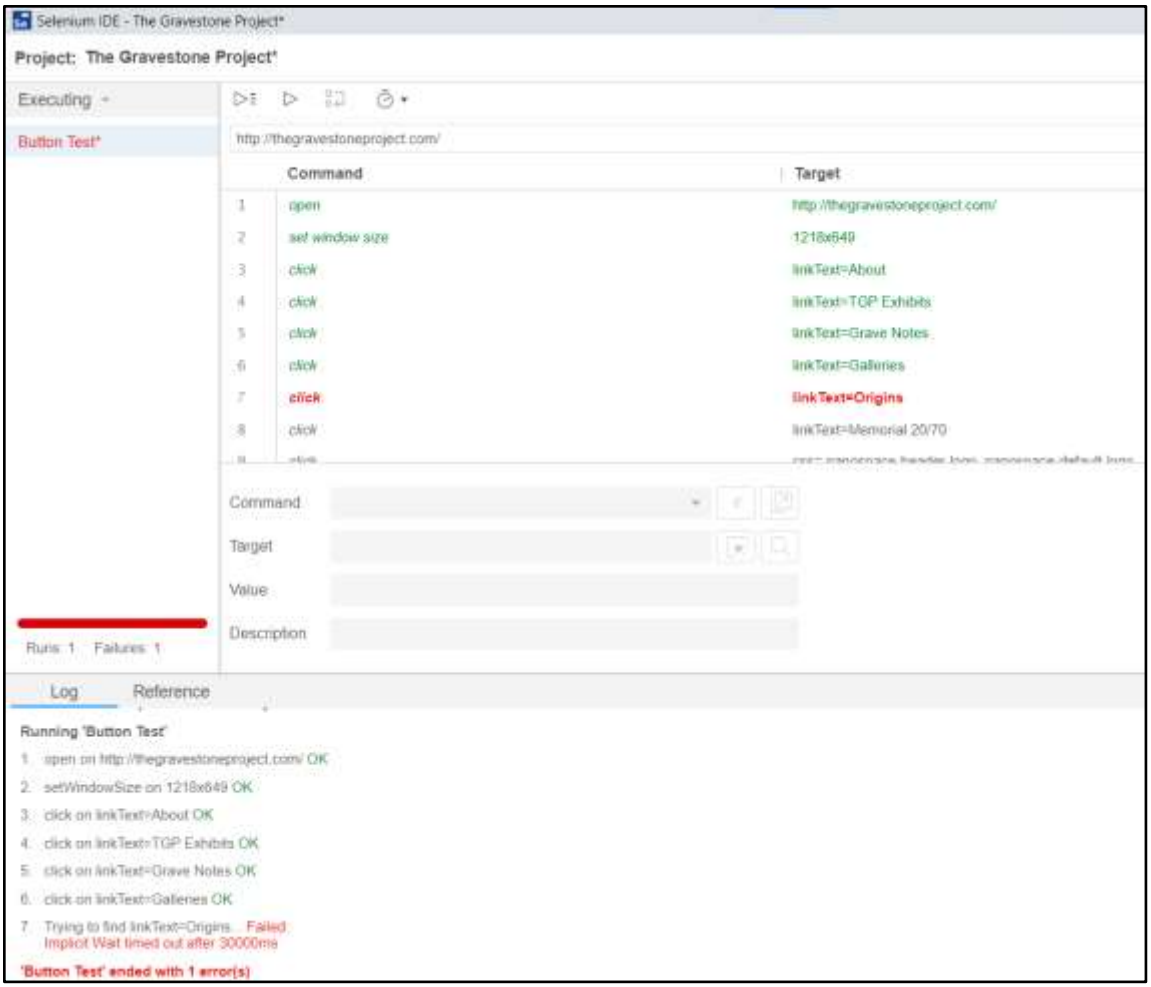


Figure 76: The Gravestone Project Timeout Error in Selenium IDE

# 8 Threat Model

## 8.1 General Website Security

Security is important for any website or application. We will be creating an SSL certificate for the site to move from HTTP to HTTPS which is much more secure as it encrypts information being sent. We will also be setting up Google Analytics to monitor traffic. If we get a ton of traffic at once from unexpected locations, we might be seeing a distributed denial of service (DDOS) attack and can take action to prevent an attacker from hosting something unwanted on our site.

Users will have logins and passwords which must be secure. We will have a meter showing how strong the password is and not allow users to sign up with weaker passwords. Passwords will need to be written twice to verify it is the same and not misspelled.

We will have a password check meeting all of the below conditions:

- Must be at least 10 characters
- Must include 1 or more uppercase characters
- Must include 1 or more lowercase characters
- Must include 1 or more symbol
- Check if password matches list of commonly used/leaked passwords [50].

If an account is spammed with password attempts that are unsuccessful, lock the account. We will decide on this number with the sponsor. The user can unlock their account through their email. In the case where a user forgets their password, we will send an email using Twilio SendGrid with a link to verify their account and change their password. We will never send a password in an email as that is a huge security risk.

We will need to monitor the amount of artifacts being uploaded as this could be a way for attackers to bring down the site by overwhelming the servers and database. Normal users will not be able to submit directly to the database, so this would involve administrator accounts only. An administrator could have a massive excel file of artifacts to upload and accidentally overwhelm the server. We could

increase our cloud server limits and database size if we see a lot of genuine user submissions. Currently, the CHDR server size is 20 TB.

We will also need to have checks that fields such as artifact description does not exceed a certain number of characters. This way we won't accidentally try to input a string longer than the size limit into the database and cause an error.

Our database will have an administrator account with a randomly generated hash. We will also be careful to avoid SQL injections. SQL injections inject malicious SQL code to view and manipulate data in a database that was not meant to be visible. This could include admin passwords, emails, and logins.

We will be hashing passwords with message-digest algorithm (MD5) on the client side when administrators sign in, sign up, and update password as well as using HTTPS.

If we have administrator users approving each other, we will need to be careful that the intended individual is actually the one who created the account. We will need email verification for this purpose where an admin will need to verify their email during signup and select a link in their email. This will trigger a change in the user table row isEmailVerified where it acts as a boolean and will change to represent true upon email verification. If we did not have an email verification check, someone could find out a team member's email and sign up with that email and pretend to be that individual and get an admin account approved without having access to the email.

We will be utilizing a few public APIs such as the Twitter API and Google Maps API. These are two companies that spend a ton of money on security, but if we use smaller known publicly available APIs we should be careful to consider their security risks and if they have had any recent data leaks.

Another issue that would need to be addressed is that the site does not automatically redirect to the https:// version of the site. This could be a big issue as without https, the site is much more vulnerable to attacks by a third party. Google separates website security into three levels:

1. **Secure:** Information sent or received through the site is secure and private. The site has a security certification and encrypts standard http connections

so a third party would not see. In Chrome this is represented by a lock next to the search bar.

2. **Info or Not Secure:** The connection to the site is not private. Some info you may enter or receive may be visible to a third party. In Chrome this is represented by a bubble with a lowercase i in it next to the search bar.
3. **Not Secure or Dangerous:** There are major issues with privacy with the site. Google further categories this section into two smaller subsections. For Not Secure, Google recommends proceeding with caution, and that people may see the information you are sending to the site. Dangerous would automatically flag the site as unsafe, giving you a warning and blocking the site from safe browsing. You can still continue on to the site after the initial block if you wish to ignore the warning. In Chrome, this is represented by a yield sign next to the search bar.

Since the site does have a proper https, that version of the site is recognized as secure. However, without the https header it is recognized as not secure (with the yield sign). Since admins use the site to log into the administrator portal, this means that if they are not cautious enough to check which version of the site they are on, they could potentially get their account information stolen by malicious attackers, which would grant them access to add malicious data to the database or desecrate the entries on it. Luckily, this is a simple problem to fix, as the solution is to simply edit the .htaccess file to force all traffic to the https version of the site.

## 8.2 DDoS Attacks

Every time a user does an action such as a search or map out artifacts, they are sending a request to the server that connects the site to the database. If too many of these requests are being sent at once, then the server may be overloaded, which would mean that the site and its services would be either noticeably slower or down, preventing users access. Realistically, the site would not have enough traffic for a denial-of-service to occur naturally, however, malicious users may attempt to attack the servers through a denial of service, or DoS attack.

A DoS may not necessarily be caused by a heavy increase in traffic, but it is one of the most common ways to cause it. A distributed denial of service, or DDoS attack is a type of DoS where multiple systems target a server at once, causing exhaustion of bandwidth of the servers until it cannot handle it anymore. Usually this is caused by multiple machines, which may or may not be caused by malware.

Unfortunately, preventing machines from being infected with or spreading malware is outside the scope of our project. There are a couple different ways to cause a DDoS attack:

**Smurf Attack:** A Smurf attack is caused by malware creating a network packet that spoofs a fake IP address. The network packet pings the server with an Internet Control Message Protocol (ICMP) message that requests a reply from the server. These replies are then echoed between the packet and server until they cause a complete denial of service.

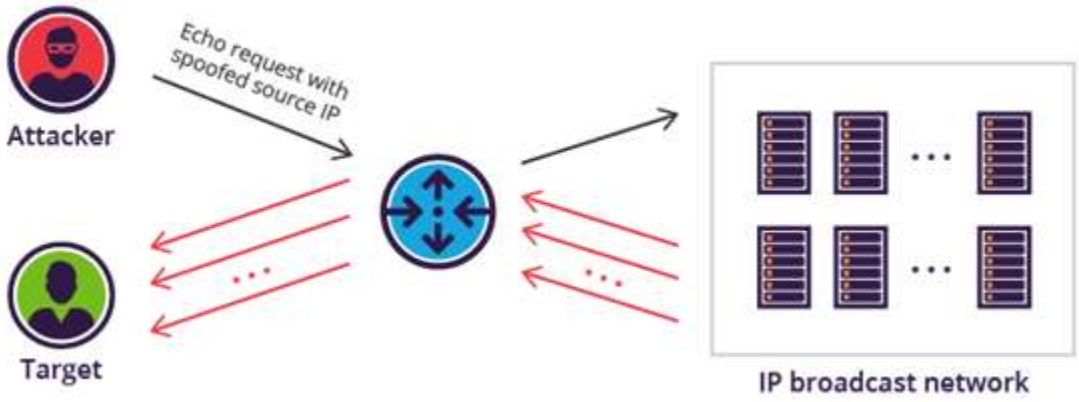


Figure 77: Diagram of a Smurf Attack [51]

**SYN Flood:** Every connection to a Transmission Control Protocol (TCP) server is described as a three-way handshake. This “handshake” starts with the client sending a SYN packet to the server as a request to connect to it. The server responds with a SYN/ACK package, in which the client then responds with an ACK package to finalize the connection. The important part of this handshake is that once the server sends the SYN/ACK packet, it will wait for the client to send the ACK packet [51].

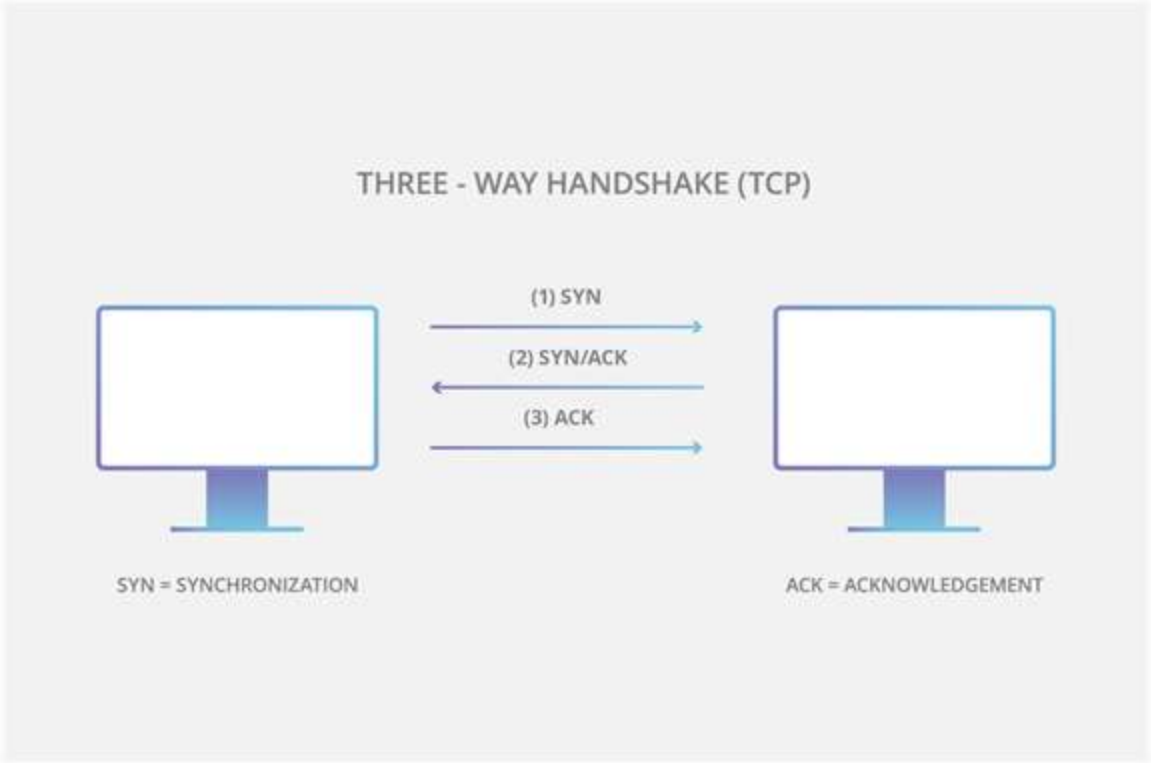


Figure 78: Diagram of a Three-Way Handshake [51]

To target the server, the attacker would exploit the fact that the server would allocate time and resources to wait for the client's responding ACK packet. The attacker would continuously send out SYN packets, which in turn would have the server open ports to wait for the client's ACK packets. However, after receiving the SYN/ACK packets, the attackers would not send out any ACK packets, leaving the allocated ports to be opened indefinitely. As ports are being held up indefinitely, the attacker keeps sending SYN packets, which in turn cause more space to be allocated for ports until the server cannot allocate any more, causing it to cease to function.

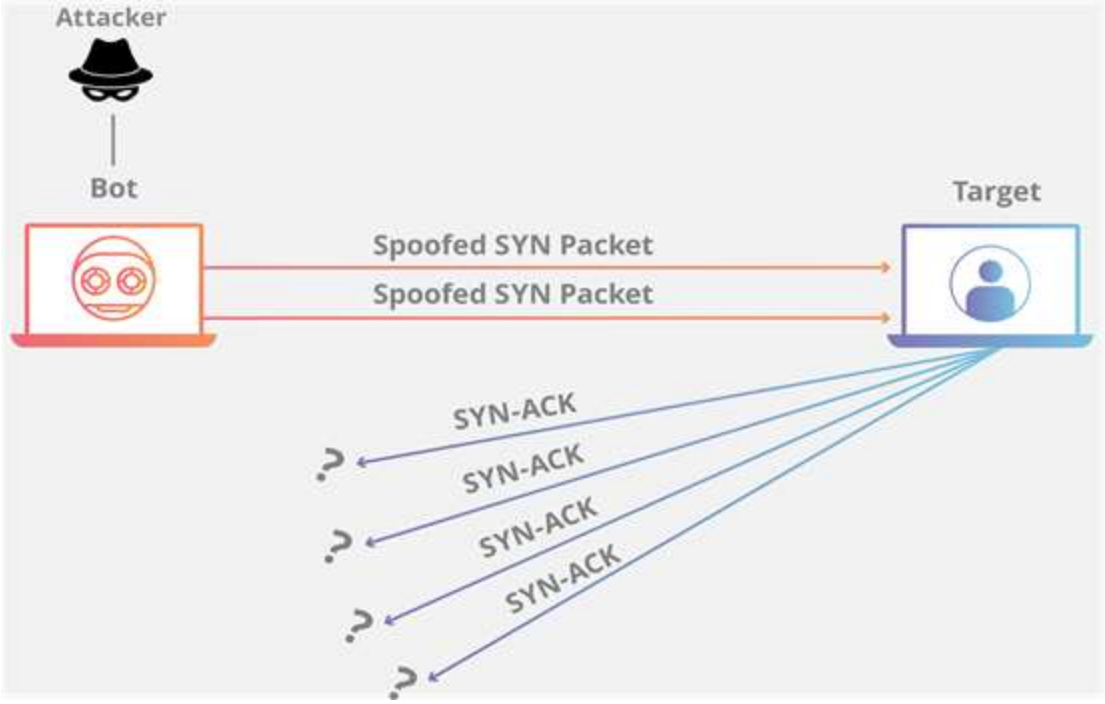

Figure 79: Diagram of a SYN Flood Attack [52]

Ideally, to deal with a DDoS attack in general, we might have to rely on a cloud provider. Cloud providers are made to withstand large amounts of requests to a server, which extend to mitigating DDoS attacks. If a cloud provider is not attainable, there are a couple ways to deal with DDoS attacks. It is very obvious to spot a DDoS as it is happening, as the app may be slower while there's an unusually high amount of traffic happening in the background. For a SYN flood specifically, we may need to increase the size of the backlog for the three-way handshake connection, and also start reusing the oldest empty TCP connection still awaiting an ACK response from the client. For a Smurf attack, we may need to monitor the attack extra closely and remove any spontaneous ICMP requests as they enter.

## 8.3 Financial Exhaustion

This project has been designed to be as cost-effective as possible. We are using UCF's C.H.D.R. server for the hosting, which is free on our sponsor's end as far as we know. Most, if not all, of our expenses will be tied to the mapping API we are utilizing. A more in-depth look at the finances and the resources we are paying for is in sections 9.1 and 9.2.

For mapping out artifacts, we are primarily using Google Maps with Tomtom and Mapbox. Tomtom and Mapbox start with free tiers, while Google Maps offers a free initial credit. Once all of those are exhausted, however, all of them start charging per certain amount of tile requests. If someone were to construct a bot to interact with the map widget on the site in a way to send an excess amount of tile requests, that would potentially cost a large sum of money. To combat this, we would simply set it that the map widget only sends one tile request at a time per IP address, and if it senses that the IP address is sending too many tile requests at a time, then it times the address out for a certain amount of time.

## 8.4 Leaks in PII

Personally identifiable information, or PII, refers to any sort of information that can be used to identify a person. Usually this includes names, contact information, and payment information. For the site that the app is being hosted on, the only PII that is being stored in the database are the names of submitters, and the email addresses and passwords of the users.

For submitters this would not be too much of an issue, as there would be nothing to leak as the names of the submitters would be public anyways, and there is not much you can do with just a name. However, if user credentials get leaked, then there could be some damage. The more pressing issue would be with the passwords if they get leaked, as that would allow unwanted people to have access to user accounts. If an admin account has their password leaked, then the perpetrator could cause damage to the site or database. Additionally, there is the issue with people who use the same password on multiple sites. While it is heavily recommended to not use the same password for multiple sites, there are people who will use the same password for every site they’re on anyways. This means that if malicious people find their password on this site, they could potentially have access to the user’s accounts on other sites. Email addresses, while not as sensitive as information as passwords, still are information that could see some repercussion from a leak. Notably, if an email has been leaked, it is susceptible to malicious attacks such as spam or phishing, putting the users of the site at greater risk of getting scammed.

As we are using UCF’s C.H.D.R. server to host our database, we will be putting our trust in it to keep everyone’s information secure. As an added precaution, we

will not be storing passwords as varchar, but rather as encrypted hash values for added security.

## 8.5 Administrator Access

If any malicious user gains administrator access by any means, as through a PII leak or otherwise, then they could deal damage to the site from any of these ways:

- Add, delete, or modify any data in the database
- Uploading malicious images or other files to the image paths
- Delete any forms submitted by users
- Mass ban or delete users
- Promote unwanted users to admin, or demote existing admins
- Invoke system maintenance to obtain backend data or to shut down access to the database

To prevent someone who isn't an admin from accessing an admin account, even if they acquired the password, we could require admin accounts to undergo two-factor authentication every time they log in. Right now, it would be as simple as just sending an email to authenticate a log-in. In the future, it may be more secure to have the two-factor authentication tied to the admin's phone number.

If any entries in the data or users list are being deleted, instead of being deleted immediately, those requests could be buffered instead. If the system is noticing that too many items are being deleted at a time, then it could cancel the buffered deletion requests and lock up the admin account until resolved by the head admin.

Promoting users or demoting admins may be not a single admin task. It may be smarter to instead have a request to the head admin to promote or demote a user instead. If not cautious enough, promoting just one admin without any sort of check may lead to a chain of mass promoting users to admin or demoting every other admin.

Uploading files may initiate another system check. To prevent potentially harmful datatypes like scripts from entering the image file paths, the system would only allow uploads of supported image types such as PNG, JPEG, GIF, etc., as well as limit to a certain file size in case unwanted admins try to upload unnecessarily large images.

# 9 Budgeting and Financing

The Gravestone Project (TGP) will not necessarily have a budget at the current time of our team working. The current plan our sponsor and co-sponsor laid out for us is that project funding will be received in the semester of Spring 2022. Their proposed uses for this funding would be to gather and provide additional information in every aspect of the project. Examples include crowdsourcing to be able to add/decode artifacts, construct a mobile application, extend artifact collection boundaries to all over the world, etc.

The expected cost of the project thus far has been most likely minimal as extensive use of University of Central Florida's tools and hosting has been excessive.

Despite this, we believe it is important to lay down the projected costs of a project like this, in case later down the line in development we need to switch services, we would have a reference to look at for pricing. In addition to costs, we will also be comparing the prices between the competing technologies of what we are using in our project. We will not be describing the pros/cons of these competing technologies in this section, only their costs. The first technology listed in each subsection of 9.1 will be our currently used technology for that subsection.

## 9.1 Project Resources

### 9.1.1 Mapping Libraries

***Leaflet**

- FREE / Open Source

**OpenLayers**

- FREE / Open Source

**Tomtom**

- Currently has a free tier.
- Free tier offers 50,000 free tile requests and 2,500 free non-tile requests daily.
- Unlimited traffic flow for mobile SDKS.
- Pay as you grow if you exceed the freemium constraints.

- .50 cents PER 1000 requests.

**MapBox**

- Currently has a free tier.
- Free tier offers 50,000 free tile requests and 25,000 for mobile SDKs.
- Pricing decreases the more active users a site has, starting from $5.00 after free tier, then decreasing by a dollar exponentially.

**Google Maps**

- Offers a free $200 credit that expires after a month, although this is not adequate for a long-term project.
- Offers MANY APIs for different cases, and each API has a different cost.
- For example, Dynamic Mapping has a cost of $7 PER 1000 requests, as well as free for mobile.

## 9.1.2 Hosting

***UCF's C.H.D.R. Server**

- Free for us, use as hosting and sandbox.

**Amazon AWS EC2**

- Currently has a free tier.
- Free tier offers up to 750 hours of uptime.
- Hourly prices range from $0.011/hour to $0.27/hour and are charged in addition to the EC2 costs.

**Azure**

- Currently DOES NOT offer a MySQL free tier.
- They offer a grand amount of price customization options.
- Hourly prices range from $0.034/hour to $5.607/hour for their highest tiers.

**Digitalocean**

- Currently DOES NOT offer a free tier.
- They give a monthly price that is estimated on your workload.
- Hourly prices range from $0.007/hour to $0.12/hour for the basic droplets.
- Premium options are available for better hardware servers.

**Google Cloud**

- Currently DOES NOT offer a MySQL free tier.

- They give a monthly price that is estimated on your workload.
- They offer a grand amount of price customization options.
- Hourly prices range from $0.021/hour to $3.23/hour for their highest tiers.

## 9.2 Total Estimated Costs

The Gravestone Projecthas no provided budget, and we're opting to utilize only tools that offer free tiers, or are free entirely. If we suddenly fine ourselves needing a paid service later on down the road, we still expect costs to remain under $100.00.

- Estimated Mapping Library Cost: $0.00
- Estimated Hosting Cost: $0.00

---

- Estimated Total Cost of Project $0.00

# 10 Project Dates and Milestones

This section lays out month-by-month the various dates and milestones pertaining to the project. Academic dates and milestones, such as the beginning and end of a semester, and included as well to paint a fuller picture of the development process in relation to the members' academia. Assignment dates are set in stone, but many of our project milestones that are not yet marked with "(complete)" may shift around as we prioritize core goals, and ensure each functionality is properly implemented.

## 10.1 May

May marks the beginning of Senior Design 1, where most of the administrative portions of the course are checked off. Groups are not yet formed, but this is the time when pitches are heard, and each student selects the project they'd like to work on.

- May 17 Summer Semester/Senior Design 1 Begins
- May 20 Pitch for The Gravestone Project Delivered
- May 27 Assignment #1: Project Selection Due (Complete)

## 10.2 June

June consists entirely of initial meetings between team members, and touching base with our sponsor to paint a clearer picture of the job that needs to be done.

- June 1 Teams Assigned, TGP Group Created
- June 2 Senior Design Bootcamp (Complete)
- June 6 Assignment #2: Bootcamp Document Due (Complete)
- June 18 Sponsor Meeting (Complete)
- June 18 Senior Design 1 Quiz 1 (Complete)
- June 20 Assignment #3: Initial Design Proposal Due (Complete)
- June 22 Senior Design 1 Project Status Check-In (Complete)

## 10.3 July

July is where we intend to do the bulk of our preparation/writing, as well as our development of a prototype of the website. Ideally, we'll "finish" all our work by the end of the month, and leave the final days of August as a buffer.

- July 2 Sponsor Meeting (Complete)
- July 9 Sponsor Meeting (Complete)
- July 14 Assignment #4: Design Document Due (Complete)
- July 23 Sponsor Meeting
- July 27 Database Configured (Complete)
- July 22 Figma Mockup/Wireframe Delivered to Sponsor (Complete)

## 10.4 August

August marks both the end of Senior Design 1, and the Beginning of Senior Design 2. We're going to slow down on work this month due to the break, but will use the free time as necessary to make up for any goals we failed to meet over the summer semester. We also each intend on spending the time between semesters to complete courses on the technologies of our respective roles.

- Aug 2 Figma Mockup/Wireframe of Website Finalized (Complete)
- Aug 5 Final Design Document Due (Complete)
- Aug 6 Sponsor Meeting
- Aug 7 Summer Semester/Senior Design 1 Ends
- Aug 23 Fall Semester/Senior Design 2 Begins
- Aug 25 Personal Courses on Relevant Technologies Finished

## 10.5 September

September is the true beginning of our development phase, and the month we intend to implement the core functionalities of the website.

- Sep 1 Sponsor Meeting
- Sep 8 Front-end home page and login page done (Complete)
- Sep 15 Website account functionality and recovery done (Complete)
- Sep 22 Webpages to submit/edit/delete artifacts done (Complete)
- Sep 29 Single artifact submission/edit/deletion done (Complete)
- Sep 29 Front-end search result page(s) done (Complete)

## 10.6 October

Development continues into October, with an added emphasis on adding the final core requirements and refining what we have already developed. Stretch goals will be in consideration this month, if we can keep up with the listed internal deadlines for core requirements.

- Oct 6 Website artifact search functionality done (Complete)
- Oct 8 Front-end for artifact mapping done (Complete)
- Oct 19 Import capability from spreadsheets done (Complete)
- Oct 26 (Stretch Goal) Import/export of metadata from site done
- Oct 26 (Stretch Goal) Interface with Voyant Tools done
- Oct 31 Confirm completion of project with sponsor

## 10.7 November

The project should be complete or nearly complete by the beginning of November, with both team members and sponsor feeling satisfied with the results. November is intended to be preparation and rehearsal for our final presentation, to ensure there are no kinks when the presentation date comes around.

- Nov 1 Preparation for final presentation begins

- Nov 8 Presentation rehearsal
- Nov 15 Presentation rehearsal
- Nov 22 Presentation rehearsal
- Late Nov Final Presentation

## 10.8 December

This is the tail end of Senior Design 2. Development will have ended before December, but this time period may contain a few loose ends to wrap up, meeting and presentation wise.

- Dec 6 Final Examination Period Begins
- Dec 11 Fall Semester/Senior Design 2 Ends

# 11 Project Summary and Conclusions

Preservation is always important in sharing various things such as knowledge, ideas, cultures, and memories. Everyone in the world has a story to them, and being able to preserve their graves and artifacts may spark fond bittersweet memories to some, and wonder to others. The Gravestone Project not only aims to simply be a tool to just view or search these artifacts, but also try to build a community out of the want to preserve history. The purpose is to not just highlight the grand or luxurious gravestones of the well-known or highly beloved, but also of the more modest gravestones of the hidden gems of the world. We hope that TGP is a project that is expanded upon after our departure from it and becomes a shining example of how much a community can accomplish to preserve a certain topic.

The team behind TGP have been working diligently over our sponsor to come up with the best idealization of this project that we can make. We divided out research and development accordingly, as sorted by our strengths and weaknesses to come up with the framework and design to go with over the scope of the project. With the tools at our disposal, we believe we have created a website that should meet all expectations, and should stand the test of time. We are lucky to have a kind and understanding sponsor like Dr. Giroux to be there to answer any questions that we may have had as well as guide us through issues and concerns with components of the project. This document, as you have seen, has gone over in detail our plans for this project, from the research to the design to the projected timeline of our project. As a team, we have given our all to research and document this project to the best of our abilities, and hope this document, and the website, are of great use to both Dr. Giroux and any future teams visiting to further development.

# 12 References

[1] statista. (2020). Ranking of the most popular database management systems worldwide, as of June 2021. Retrieved from https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

[2] Mysqltutorial. (2021). MySQL Primary Key. Retrieved from https://www.mysqltutorial.org/mysql-primary-key/

[3] MySQLTutorial. (2021). Import CSV File Into MySQL Table. Retrieved from https://www.mysqltutorial.org/import-csv-file-mysql-table/

[4] stateofjs. (2020). Front-end Frameworks. Retrieved from https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/

[5] The Gravestone Project. (2021). Culture / Literature / Memorialization. Retrieved from https://thegravestoneproject.com/

[6] The Gravestone Project. (2021). Memorial 20/70 Phases. Retrieved from http://thegravestoneproject.com/memorial-20-70/memorial-20-70-phases/

[7] The Gravestone Project. (2021). Grave Notes. Retrieved from http://thegravestoneproject.com/grave-notes/

[8] The Gravestone Project. (2021). Cemeteries. Retrieved from http://thegravestoneproject.com/galleries/cemeteries/

[9] The Gravestone Project. (2021). Notable Graves. Retrieved from http://thegravestoneproject.com/galleries/notable-graves/

[10] The Gravestone Project. (2021). Methodology. Retrieved from http://thegravestoneproject.com/about-the-project/methodology/

[11] The Gravestone Project. (2021). Project Origins. Retrieved from http://thegravestoneproject.com/about-the-project/origins/

[12] The Gravestone Project. (2021). Project Collaborators. Retrieved from http://thegravestoneproject.com/about-the-project/project-collaborators/

[13] Kranz, Garry. (2021). What is Metadata and How Does it Work? Retrieved from https://whatis.techtarget.com/definition/metadata

[14] W3Schools. (2021). HTML <meta> Tag. Retrieved from https://www.w3schools.com/tags/tag_meta.asp

[15] MerlineOne. (2021). What are the Different Types of Metadata (and How are They Used)? Retrieved from https://merlinone.com/types-of-metadata/

[16] National Microbiome Data Collective (NMDC). (2021). Introduction to Metadata and Ontologies: Everything You Always Wanted to Know About Metadata and Ontologies (But Were Afraid to Ask). Retrieved from https://microbiomedata.org/introduction-to-metadata-and-ontologies/

[17] GitHub. (2021). react-ga (React Google Analytics Module). Retrieved from https://github.com/react-ga/react-ga

[18] Emil Drkušić. (2016). All About Indexes Part 2: MySQL Index Structure and Performance. Retrieved from https://www.vertabelo.com/blog/all-about-indexes-part-2-mysql-index-structure-and-performance/#:~:text=MySQL%20uses%20both%20BTREE%20(B,use%20both%20HASH%20and%20BTREE

[19] Arnab Roy Chowdhury. (2021). 21 Best React Component Libraries To Try In 2021. Retrieved from https://www.lambdatest.com/blog

/best-react-component-libraries-2021/

[20] Webucator. (2021). How to Send and Receive JSON Data to and from the Server. Retrieved from https://www.webucator.com/article/how-to-send-and-receive-json-data-to-and-from-the/

[21] Digital Ocean. (2021). How To Use JSON.parse() and JSON.stringify(). Retrieved from https://www.digitalocean.com/community/tutorials/js-json-parse-stringify

[22] stackshare. (2021). Alternatives to Leaflet. Retrieved from https://stackshare.io/leaflet/alternatives

[23] Smashing Magazine. (2020). How To Create Maps With React And Leaflet. Retrieved from https://www.smashingmagazine.com/2020/02/javascript-maps-react-leaflet/

[24] Paul Le Cam. (2021). Introduction. Retrieved from https://react-leaflet.js.org/docs/start-introduction/

[25] Leaflet (2021). Extending 2 Layers. Retrieved from https://leafletjs.com/examples/extending/extending-2-layers.html

[26] Leaflet. (2012). Leaflet.MarkerCluster 0.1 Released. Retrieved from https://leafletjs.com/2012/08/20/guest-post-markerclusterer-0-1-released.html

[27] Zooniverse. (2021). About. Retrieved from https://www.zooniverse.org/about

[28] Voyant-Tools. (2021). Cirrus. Retrieved from https://voyant-tools.org/docs/#!/guide/cirrus

[29] Nathan Sebhastian. (2021). Wouter: A Minimalist Alternative to React

Router. Retrieved from https://blog.bitsrc.io/wouter-a-minimalist-alternative-to-react-router-2756690c2b77

[30] Ryan Florence. (2019). The Future of React Router and @reach/router. Retrieved from https://reacttraining.com/blog/reach-react-router-future/

[31] Medium (2020). Wouter — A Minimalist 1.2kb Alternative to React Router. Retrieved from https://medium.com/habilelabs/wouter-a-minimalist-1-2kb-alternative-to-react-router-f6145f3b4b0e

[32] Spec. (2021). The 8-Point Grid. Retrieved from https://spec.fm/specifics/8-pt-grid

[33] Medium. (2019). The Comprehensive 8pt Grid Guide. Retrieved from https://medium.com/swlh/the-comprehensive-8pt-grid-guide-aa16ff402179

[34] Dominic Fraser. (2018). Testing React with Jest and Enzyme I. Retrieved from https://medium.com/codeclan/testing-react-with-jest-and-enzyme-20505fec4675

[35] Testim. (2021). What Is the Difference Between Jest and Enzyme? Retrieved from https://www.testim.io/blog/what-is-the-difference-between-jest-and-enzyme/

[36] Google. (2021). Search Results for “cemeteries”. Retrieved from https://www.google.com/maps/search/cemeteries/@27.1127713,-80.4344457,10z/data=!3m1!4b1

[37] Find A Grave. (2021). Gravestone for Ben Ali Néhari El Methuani Sabah. Retrieved from https://www.findagrave.com/memorial/223831108/ben_ali_n%C3%A9hari-el_methuani-sabah.

[38] Target. (2021). Search filters for Target's apparel section. Retrieved from https://www.target.com/s?searchTerm=pants&Nao=0

[39] Material UI. (2021). Material UI editions, plans, and pricing. Retrieved from https://next.material-ui.com/branding/pricing/

[40] Bootstrap. (2021). Bootstrap License FAQ. Retrieved from https://getbootstrap.com/docs/4.0/about/license/

[41] ASAP Developers. (2018). React License: React Native update to MIT License. Retrieved from https://www.asapdevelopers.com/react-native-update-to-mit-license/

[42] MDN Web Docs. (2021). Introduction to Web APIs. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/introduction

[43] RestfulAPI. (2021). HTTP Status Codes. Retrieved from https://restfulapi.net/http-status-codes/

[44] Jestjs.io. (2021). Getting Started. Retrieved from https://jestjs.io/docs/getting-started

[45] Jestjs.io. (2021). Testing Asynchronous Code. Retrieved from https://jestjs.io/docs/asynchronous

[46] BrowserStack. (2021). Selenium WebDriver Tutorial: Getting Started with Test Automation. Retrieved from https://www.browserstack.com/guide/selenium-webdriver-tutorial

[47] Harita Ravindranath. (2021). How To Use Strings In JavaScript With Selenium WebDriver? Retrieved from https://www.lambdatest.com/blog/using-strings-in-javascript-using-selenium-webdriver/

[48] Github. (2017). harishyadav15 / NodeJS-Login-Test. Retrieved from https://github.com/harishyadav15/NodeJS-Login-Test/blob/master/logintest.js

[49] Selenium IDE. (2021). Getting Started. Retrieved from https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started

[50] cybernews. (2021). Has your password leaked? Retrieved from https://cybernews.com/password-leak-check/

[51] imperva. (2021). Smurf DDoS attack. Retrieved from https://www.imperva.com/learn/ddos/smurf-attack-ddos/

[52] Cloudflare. (2021). SYN flood attack. Retrieved from https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/