Mapping Keats

Students: Joshua Colston, Ryan Philbin, Tyler Goldberg, Thanah Raveendran, Ege Topcuoglu
Sponsors: Dr. Amy Giroux, Connie Harper, Brook Miller, Brian Rejack

Contents

1 Executive Summary

The *Mapping Keats* project is a web application that visualizes all 252 of John Keats' famous letters in a variety of forms. It was designed and built by Joshua Colston, Tyler Goldberg, Ryan Philbin, Thanah Raveendran, and Ege Topcuoglu. There is a great deal of information and blog posts about Keats and his letters, as well as the themes that carry over from one letter to another. The goal for this project is to expand upon the information already out there, and create an interactive website that is fun and informative to use so that users will want to learn more about Keats and his letters while staying useful to researchers with their topic of observations.

When a user first visits the website, they will be greeted by a heat map that shows what parts of the world Keats sent the most letters from. It can also display a similar heat map, where recipients' locations will take over the circles of the map. A query system will enable the user to search for any number of letters using multiple fields including but not limited to: date, location, common themes in the letter, relevant tags, a text search, and more. These results can be viewed in a list format, as well as displayed on the map. Results can be further filtered by time and location, including the selection of a place and a given diameter around that location.

Another key feature for this application is to provide the user with a temporal slider, allowing them to scroll through the time period when these letters were sent, and watch the letters fly from their original location to wherever their recipient is.

Since this project is all about visualizing pre-existing data, we have put lots of thought into determining the best way to provide it to the user. Leaflet is a javascript framework that provides a plethora of different options for displaying maps and plotting relevant information, which is perfect for our project and was actually required by our sponsors. Once we determine the interface that best fits the user's needs, Leaflet gives us the freedom and ability to build it to our specification.

This document details how the team designed and built this project, including any research, design tools, management information, project requirements and goals, and any other relevant information.

2 Overview

2.1 Project Narrative Description

The goal of this project is to provide an interactive map so that users can explore and read all 252 of John Keats letters that he sent and received. This website will utilize a heat map to show where Keats sent and received the most letters, as well as display a subset of letters from the original 252, according to a user query. Users can search letters by date, location, themes, tags, or even search the text of the transcription for each letter. By providing an easy to use application which allows users to quickly access information about any one of these letters, we can promote the learning of John Keats' story and legend. These letters give us the ability to look into the thoughts of one of the most popular poets of the 19th century.

For the website, we will build the frontend with React.JS, and use Node.JS and Express.JS for the backend, along with a MySQL database. Our project will be hosted on the UCF Center for Humanities and Digital Research (CHDR)[1] web server with the option of going public in the future. We will use Leaflet, a JavaScript library, to create the heat map and any other map designs. In addition to allowing the user to search for specific letters, our goal is to provide a temporal slider for the heat map so that the user can scroll through time, and watch the letters travel around the world. We may have to create a heat map to show which cities had the most sent and received letters, as well as a different style of map to show the direct paths that each letter took. Mapping all these paths on top of the heat map may overstimulate the user and just look incoherent.

2.2 Statement of Motivation

Our team is excited to tackle the problem presented to us for this project, as we have ranked all the projects and put this one near the top of our lists. All our members have some experience in website development, but are eager to learn more and put our skills together to make the best final product that we can. We are going to take this project very seriously as our Senior Design Project is probably going to be the most complex and, in the end, most interesting and complete product that any of us have worked on.

We have taken the core requirements given to us by our sponsor, Dr. Amy Giroux, and started thinking of how we will implement this in the near future. Along with the stretch goals given to us by Dr. Giroux, it is fascinating to see that we are developing our own ideas for features to try and implement, which we will propose to the sponsors to see if they would like it added.

2.2.1 Joshua Colston

I have always found an interest in project management when developing full stack applications. In COP4331 I was the project manager and tried to empower everyone in the group by giving them the freedom to design the app the way they wanted while maintaining the overall vision. However, at times I felt my project management skills weren't up to the bar when it came to doing some of the administrative work such as Gantt Charts, Agile workflow, and staying on task during meetings.

Mapping Keats is a great way for me to not only expand my knowledge about John Keats letters, but another way for me to improve not only as a developer. Day one, I had a big interest in this project as I could see the broader impacts it could have on those who wish to learn about John Keat. I truly believe this product will provide a centralized application for everyone to learn more about his past.

2.2.2 Ege Topcuoglu

After listening to more than thirty project pitches in the class, both from sponsors and from my classmates, it was really hard to list ten good projects that I want to be part of. My personal criteria for listing a project as my top ten were, how fun the project is, how established is the idea, how involved are the sponsors, how much support do they give and probably most importantly, do I want to participate in creating a project or solving a problem like that proposed in the classroom.

Mapping Keats project caught my attention from day one. My primary motivation for listing this project in my top ten list was that it fit all my criteria. First of all, tracking how letters of John Keats' traveled through the 1800's sounds incredibly cool. And being able to see the letters in an interactive map where we can track destinations, see heat

maps and actually being able to read them sounded extremely fun to implement. Then I saw how established the idea is. Dr. Giroux, who is one of our sponsors, has sponsored many Senior Design projects, and this project is well thought out, and pitched in a very concise and effective manner. After listening to this pitch, I could already imagine what parts are needed for this project and were thinking about ideas for the implementation. And clearly Dr Giroux is very involved with Senior Design projects as she sponsors four other projects just for this semester. And lastly, Mapping Keats project offered a lot of support as Connie Harper, a UCF Software Developer is offering full support to guide us through the project as needed. We will also be working alongside with Brian Rejack from Illinois State University who is an expert in this field. Seeing this kind of support and resources made me think about this project a lot.

Overall, Mapping Keats project was really interesting to me from day one, as it looked fun, the idea was clear, sponsors were involved in the pitch, and the support available for this project was solid. These criterias made me want to tackle this project and that is what motivated me to put this project in my top ten list.

2.2.3 Tyler Goldberg

While I am going for my computer science degree, I've always held a great love towards many fields originally outside of the tech industry. Not all CS work is about pushing the technological field to the next level, a lot of the time it's about using modern tools to make old fashioned work easier and more available. This project encompasses one of my core values of effectiveness by enabling research that would normally have to be done manually and almost impossibly and combining it with modern technology. I enjoy doing good work that benefits a niche so they can accomplish their goals. This project is important to someone's research, and I feel honored to see their passion accomplished.

I don't know if I would believe anyone who says they aren't taking this class to graduate; that holds true for me as well. I am excited to get through this class and enter the same workforce my family is in. This project holds experience on several topics in the technological spectrum, from database work to website building. I am not the most experienced with such development and am both excited and nervous to learn these new things.

2.2.4 Thanah Raveendran

At face value, this project appeared straightforward and ultimately limited in scope. That being said, I realized the potential that could be explored with it. Given the relatively simple functionality of the application, it gives me more room to take risks and explore the technologies we use. My previous experience with most of these tools and frameworks have only been within the confines of course requirements, so I'm enthusiastic to explore their further applications and simply practice using them.

Outside of coding, I'm hoping to develop good work practices and treat the progression of this project as I would a project in a professional environment. I have worked with project organization tools before, such as Gantt charts and Agile, and hopefully I can be introduced to more of these utilities through my groupmates. I have previous experience with project management in a different field, so I'm fascinated in how I can translate some of those skills into a software development project and benefit from the smooth progression of the workflow.

2.2.5 Ryan Philbin

When this project was presented to our class, I immediately was attracted to the idea of learning another new way of presenting data to the user. I have never built a heat map before but I have always enjoyed using and looking at them, especially when they are interactive. Web App Development has been a skill that I have been practicing both in school and on my own time for the past 7-8 months, so learning an additional skill inside of that field seemed like a great learning opportunity and even a bit of a challenge.

Another part of this project that intrigued me was the fact that we have our main sponsor for the project, as well as a couple of professional developers that are there to give us some assistance when needed. One of the main requirements for this project is that we use Leaflet, a JavaScript library, and we were connected with Brook Miller who has experience with Leaflet, and will be able to point us in the right direction if we are stuck. I also think talking with these professionals will give us some insight into what programming for a real company would be like, since we need to deploy our web app to their servers and learn to use their domain for our website.

2.3 Broader Impacts

Research forums and sites to learn more about John Keats letters are across the globe but this site will be one of the first to plot his letters on a historical map. Visualizing the meta data using technology such as Leaflet.js gives the user the full knowledge of things such as location, time periods, and seeing the letter deliveries over time. With this web application, we hope to expand the research and analysis of John Keat for our education systems. With our stretch goals, we hope this full stack web application can be applied to anyone's writings (e.g. letters of Ben Franklin) along with giving the user the ability to save their queries, discussion posts, and notes on these letters as they analyze them on the site. Overall, the broader impacts of this product involve not only educating others but giving users a way to visualize letters of John Keat.

2.4 Legal, Ethical, and Privacy Issues

All of John Keats letters are currently open for use to the public. Currently, there are no websites that map, display, or query John Keats letters and poems. We believe that there will be no privacy issues as we aren't collecting any detailed user data along with this product being available to the public at no cost. The only user data that is currently going to be stored are admin accounts, which help perform CRUD operations on the database, and help to maintain and improve this product. In the near future, our customer mentioned adding user accounts to save and access their queries but we do not see any legal or ethical issues with this stretch goal.

2.5 Goals and Constraints

This section will describe the main goals for this project, as well as provide a list of constraints. These goals are not detailed requirements, but a guide to help understand the overall functionality of the website.

2.5.1 Goals

- Display John Keats' collection of letters by using a heat map, but also have the ability to display one letter's journey across Europe on a map.
- Allow the user to search for one or more letters by entering multiple fields. Then display those results on the map.
- Animate letters flying across the map as the user scrolls through time using a temporal slider

2.5.2 Constraints

- Project files will be hosted on the UCF CHDR server, and should not require 3rd-party services for functionality.
- This website must utilize a MySQL database so that it can be encapsulated by the host server.
- Leaflet.js must be used to display period maps, letters, heat circles, city markers, and other relevant information.

2.6 John Keats & His Legacy

John Keats was a famous poet in the early 19th century, born in London, England. His works received lots of disapproval from critics during his life, yet were perceived as very influential to the English writing culture, after his death in 1821. His works were known for containing vivid imagery and topics that garnered lots of appeal from his audience. Like many famous artists or writers, his work was not appreciated much during his lifetime, although they were only published for a handful of years before his untimely death. He died at the very young age of 25, which did bring interest towards who he was as a poet, thus exposing his miraculous works of literature to the world. The style he wrote some of his poems in, especially his *1819 Ode's*, a type of lyrical stanza, were some of the last works he ever wrote and also some of the most influential writing to future English poets. [2]

2.6.1 John Keats' Letters

His series of letters were published in the mid 1800's and became quite famous in the 1900's, being studied by academics all over the world. They give readers an inside look into Keats' mind and inspiration for his poetry, along with his deepest desires as to what purpose he wanted his literature to serve. These letters are written to many people, some containing jokes between friends, opinions about current events, early drafts of poems, and deep thoughts about his human nature. Robert Gittings, an English writer and biographer in the 20th century, said that he sees these letters as a spiritual journey, not just for Keats but for any person to read and learn from.

Readers and writers alike can learn so much about the true beauty surrounding the creative process that is writing poetry. Keats likes to write in these letters about the nature of why he is writing poetry, his deepest thoughts about life and love, and his always evolving philosophy of the world. Many of his letters between other poets at the time are written with a style that mimics a poem, using beautiful language to express well thought out ideas. Some of his letters also share a few common themes and concepts like "the mansion of many apartments" and the "chameleon poet," which are explained by Keats to his recipient.[3] Another reason why his letters became so popular was probably because since he had only 54 poems published, his series of 252

letters was a larger body of work for people to read and study. It could be argued that his letters give much more insight into what kind of writer he was, more than his poems, since there are less of them.

2.7 Consultants

During our meeting with our customer. We were introduced to some consultants for this project. Some who are experts using libraries such as Leaflet.js and development on the UCF CHDR server.

-Amy Giroux, PhD. Center for Humanities and Digital Research. Primary sponsor and liaison to Brian Rejack for whom this project will be benefiting.

-Brian Rejack. Illinois State University. Department of English. Will help by providing project requirements as he will be the primary user.

-Connie Harper. Software developer for CHDR. Will assist with getting our project up and running on the CHDR server and any problems we encounter along the way.

-Brook Miller, PhD. Applications Programmer with experience with Leaflet that will assist with its learning curve. Also has a PhD in English with a good sense of uses that research scholars have with projects like this.

3 Requirements and Specifications

3.1 Admin System

For this website, we will create an admin portal that allows a user, who has been promoted to an admin, to access certain functions and information that is necessary for upkeep of the website. This is an essential part, as it allows admins to perform simple operations on the database without having to open a MySQL console, and type in complicated commands in order to add a new letter, edit an existing one, or even remove it. Making this portal easy to use is essential, as we want it to feel intuitive for the user to navigate to the correct page in order to accomplish whatever task they need to complete.

3.1.1 Admin System Requirements

- An Admin portal that can execute CRUD operations on the database. CRUD stands for Create, Read, Update, and Delete.
- Ability to promote another user to Admin level permissions.
- Ability to demote another user back to User level permissions.
- Admin Login Page that supports “forgot my password” functionality.
- Before an Admin adds a new Letter to the database, have them confirm that information is inserted into the proper columns and maintains proper format.
- When searching for a letter, return results in a card format for easy viewing.
- When results are returned, include a thumbnail for each letter.
- Ability to take in any date format from the user and correctly store it in the database.

3.1.2 Admin System Stretch Goals

- Ability to take in an excel spreadsheet and parse the contents in order to create new letters to insert into the database.
- Example Template for an excel spreadsheet that would be uploaded to create letters.
- Add profiles for People so that users can track recipients of letters easier and see more information about important people related to Keats and his letters.
- The ability to take this full stack web application and apply other authors/poets into this application for research and data visualization purposes.
- The ability for an admin to create a new author to add/store their writings for users to see. (e.g. ability to see John Keats and Ben Franklin's letters on the same map)

3.2 User System

The User system is the main section of our website, and 99% of users will only ever interact with this system. This means we must create a pleasant user experience on these pages that has a multitude of functions for the user to play and experiment with, as well as a fluid method of interacting with the map. Since the map is the main screen users will utilize, making sure anything they need is just a click or two away is very important to keep in mind when designing this system.

3.2.1 User System Requirements

- An interactive map that allows users to view where John Keats wrote and sent his famous letters.
- Allow the user to choose from a variety of map overlays, including but not limited to: heat map, individual letter map, etc.

- Each letter should be able, on selection, to draw a line on the map from the origin to the destination.
- Display an image of each original letter along with a transcript for the user.
- On selection, each letter will display all key information to the user, as well as provide tabs for an image of the original letter and its transcript.
- Provide a temporal slider to allow the user to slide through time and watch the letters travel across the map.

3.2.2 User System Stretch Goals

- User portal where users can sign up and save their existing queries.
- Adding profiles for recipients of letters. This would help the user visualize just how many letters were sent to certain individuals, and view those letters as a collection.
- Mobile friendly User Interface.

4 Division of Labor

During our initial meetings with the group. It was crucial to assess everyone’s strengths and weaknesses to determine what was the most viable role for everyone. Overall, a lot of us seemed to have great experience with server-side development. However, some wanted to gain experience with front-end development. In this section, we will cover our group's division of labor for this project.

4.1 Ege Topcuoglu

Ege’s initial interest was to be on the back-end team for this project. In his previous projects in the University of Central Florida, in the Processes for Object-Oriented Software Development course, he worked in two projects. In the first project, he did the database design and implementation such as creating tables, keys and relationships between tables in a relational database SQL. In his second project, he again implemented a database this time using a non-relational database with MongoDB. Addition to that, he also implemented an API for CRUD operations in React using JavaScript. After that, he implemented authentication and verification security using JSON Web Tokens (JWT) and he implemented Forgot Password and Reset Password functionality using NodeMailer and again JSON Web Tokens. In his Database Management Systems class, he also worked in a project doing backend using PHP where he created functional webpages.

So naturally, his first interest was in the back-end for this project. But after a few meetings with his team, seeing that everybody in his group was open to try new things and learn, he changed his mind and decided to be in the front-end team for this project. After working in the back-end for the last three projects, he wanted to get experience in the front-end to be fluent in full stack development. Since we all need to know and learn new technologies and adapt as a computer scientist, Ege thought Senior Design 1 is a great opportunity to learn front-end development since in Senior Design 1 we have time to research, study and learn in’s and out’s of the new technologies.

At the beginning stage of the project Ege communicated with his front-end members, and after the group decided on using React, he decided to take part in researching it,

and learn everything about it. He researched advantages and disadvantages of React, compared React to other solutions like Angular and Vue and researched most used libraries like Fluent UI and Material UI and Bootstrap. In addition to that he learned about good coding practices in React like components and props as well as routing. He also researched the difference between declarative programming and imperative programming.

After that he researched data fetching for the front-end with fetch() and Axios in React by learning about their advantages and disadvantages and comparing them. He also made contributions to the wireframes and created the initial wireframe using figma designing tool.

4.2 Joshua Colston

Josh’s initial interest was taking over the project management role along with implementing the database for the project. In Processes for Object-Oriented Software, he was also project manager along with developing the API routes.

Naturally, Josh was the one to take initiative to create the discord server along with the Trello Board. He then delegated some additional roles and responsibilities to everyone such as inquiring everyone’s strengths/weaknesses along with what everyone wanted to do during this project.

From there, he decided to work on the database and API roles, since that is where he was most successful in past projects. He researched the different relational databases for his customers such as PostgreSQL and MySQL to see which was best for this project. During this process, he also considered other constraints such as budgeting, licensing, and open source technologies that would be most viable to fulfill his customers' needs.

Moving forward, Josh has completed the database schema and began to work with Thanah on creating some of the admin API routes such as login/signup. Next semester, he will transition his focus to the API routes along with making sure the prototyping and testing is successful as development continues.

4.3 Ryan Philbin

Before working on this project, Ryan did not have any experience in designing websites using React. He had built some much simpler websites using HTML and CSS, but wanted to face a new challenge considering this is a Senior Design project. Ryan has built quite a few API endpoints using Node and Express, and did not want to fall back on what he is already comfortable with.

For this project, Ryan's main area of focus was building and designing the look and feel of the website with the other members of the front-end team. He worked with the other members to gather the wants and needs for the website from our sponsor, and got to work designing wireframes to present to our sponsors. After multiple rounds of presenting these wireframes and taking critiques and other requests, they finally came to a final interface and began to start building their vision.

Ryan did research on John Keats and his letters, the wireframe software Figma, and Express.js. He also described how the query system would work and how data would be visualized on the front end. He made an executive summary for the design document, a sequence diagram, a use case diagram, and some wireframes.

Moving forward, most of Ryan's labor on this project was spent on building the website using React, and connecting it to the endpoints, collaborating with the back-end team. Building the user interface required lots of work, and so did constantly testing features and trying to break the website to find bugs and determine where improvements needed to be made. Another tricky feature to implement was the temporal slider, and required lots of time and careful thought.

4.4 Tyler Goldberg

Before this project, Tyler had near to no experience with developing a website. In the past, he had done some database work using Google's Firebase for an android application that allowed users to play Poker together, but nothing on the scale this project would require. With a strong willingness to learn and a solid team to support, he gained the confidence to work with the other members. As the group was having their first meeting discussing strengths and weaknesses and who would be doing what, Tyler volunteered to work on the front end of the project. He would not yet have the expertise

required for the role, but Senior Design is all about removing oneself from their comfort zone and gaining new skills.

With roles now determined, Tyler got to work researching Leaflet. As this was the tool the project would be using to display and map all its data, this task would be imperative for future success. First building a foundation of what Leaflet is and how best it can be utilized for our purposes compared to other software, Tyler then moved on to figuring out the intricacies of the program for future use.

Moving on from research, the next major task Tyler worked on was creating wireframes. The front-end developers decided to use Figma for their conceptualization. Each dev would initially work separately to bring to life an idea for what the webapp will look like, then coming together to discuss and compare. While there were many similarities, Tyler was able to add all the best from each frame together for a project worth presenting to the sponsor. After showing the sponsor the initial idea, they had many comments about it. Tyler oversaw that these changes made it to the wireframes until everyone was happy with them.

Tyler Goldberg looks forward to next semester where he will work together with his team to develop a quality product. He is excited to begin transferring the ideas from the wireframes to a realized webpage including all the work needed to get the mapping set up and working.

4.5 Thanah Raveendran

Thanah initially had minimal experience working in full stack projects, so he aimed to partake in a project that would test his ability to operate in the fundamental mechanisms of a full stack application. As such he took on the role of primarily aiding in the development of the API for this full stack web application, with the intention of aiding in frontend development as the project further progressed.

As such, he took it upon himself to research some of the tools available to develop the API, settling on Node and Express as well as Sequelize after comparing several alternatives. The final choice of NodeJS was settled on after coming to a compromise with the customer on developing an API that would work with the CHDR server they had provided the group.

After researching the framework that would be most appropriate for the project and group, he began working with the project manager, Joshua Colston, in outlining a tentative timeline for the design and development process of the project. Once this had been finalized and progress on the design document had been made, they two began API development.

To allow basic frontend development to begin, Joshua and Thanah began developing some of the foundational API endpoints to allow frontend to begin working with the server. Of these, they particularly focused on the login function, the create account function, the change admin level function, and the delete user function.

5 Research

5.1 Front End Research

In our initial meeting about ideas and implementation and after a brief talk about front-end and what to use in our Senior Design project, we have decided to use a JavaScript library. None of us had an experience with using Python for front-end development, and none of us wanted to use PHP. We have decided on React as our JavaScript library since we were a little familiar with React from previous classes and that experience is valuable. We believe React is the optimal solution to our front-end development. Below is the explanation of what React is, advantages and disadvantages of using React, and a comparison of React to other solutions.

5.1.1 React

React is a JavaScript library to build User Interfaces based on UI components. It is created and maintained by Meta, formerly known as Facebook[4]. It is one of the most popular JavaScript libraries out there in the market. It is open-source and free. React also makes use of JSX which is an extension of JavaScript that looks very similar to HTML while incorporating JavaScript functionality. Here is a snippet of code in React with JSX in Figure 5.1.1.A.

```
// JSX
const page = (
    <div>
        <h1 className="header">This is JSX</h1>
        <p>This is a paragraph</p>
    </div>
)
ReactDOM.render(
    page,
    document.getElementById("root")
)
```

Figure 5.1.1.A Sample JSX Code.

React also takes advantage of states and components for efficient and fast front-end development and has its own Virtual DOM that provides a massive boost to the performance as this DOM allows ‘reactive’ development. React also uses declarative programming paradigm instead of imperative programming.

Declarative vs Imperative Programming

Imperative programming is what we know from normal JavaScript. In imperative programming we follow each step one after another to reach our end goal. We have to follow each procedure or the program will crash. It’s more sequence driven and it’s used greatly where we need a sequence of events to implement.

Declarative programming is driven by results. Instead of sequences we focus on the end product and describe that. That often means disregarding the steps. One great example to describe the difference between two is an address example[5]. In an imperative example we would describe how to get to that address step by step, describing roads, turns and how far we want to go. Declarative version of that would be saying that place is located in that address, with maybe an addition of a couple more streets, that’s all.

React uses declarative programming. Even though it’s different from what we know normally and that takes effort to learn and improve on, it makes things very simple to just ‘declare’ what we want to be shown on screen and not worry about what happens behind the scenes. Here is what declarative and imperative programming looks like on React versus pure JavaScript and how they compare to each other in code in Figure 5.1.1.B down below.

```
// Declarative
ReactDOM.render(<h1>Hello World!</h1>, document.getElementById("root"))


// Imperative
const h1 = document.createElement("h1")
h1.textContent = "Hello World!"
h1.className = "header"
document.getElementById("root").append(h1)
```

Figure 5.1.1.B. Difference Between Declarative and Imperative

Of course, this example is just to show how both paradigms work and doesn't reflect how we really do things but, still we can see how in declarative way we just describe what we want rather than imperative way that has sequences of code that follow each other. Declarative programming is just a simpler and more elegant solution for User interface implementation.

Components and Props in React

One of the main reasons we want to use React, other than familiarity from previous classes, is React offers great compactness and reusability in the form of components and props. With components and props our code becomes so much cleaner, readable, and intuitive.

Components

Components in React are basically reusable pieces of code that can correspond to parts of the User Interface like navigation bar, footer, etc[6]. These components usually get coded in JavaScript functions, and they can have their own files. Therefore, after we build up those components, we can then reuse those files however we like while making changes to those components becomes so much easier. And at the end, we can even go further and put those components in a component itself and render that component as a whole component by itself. Here is a very basic example of that idea in React and how it works in code in Figures 5.1.1.C, 5.1.1.D and 5.1.1.E.

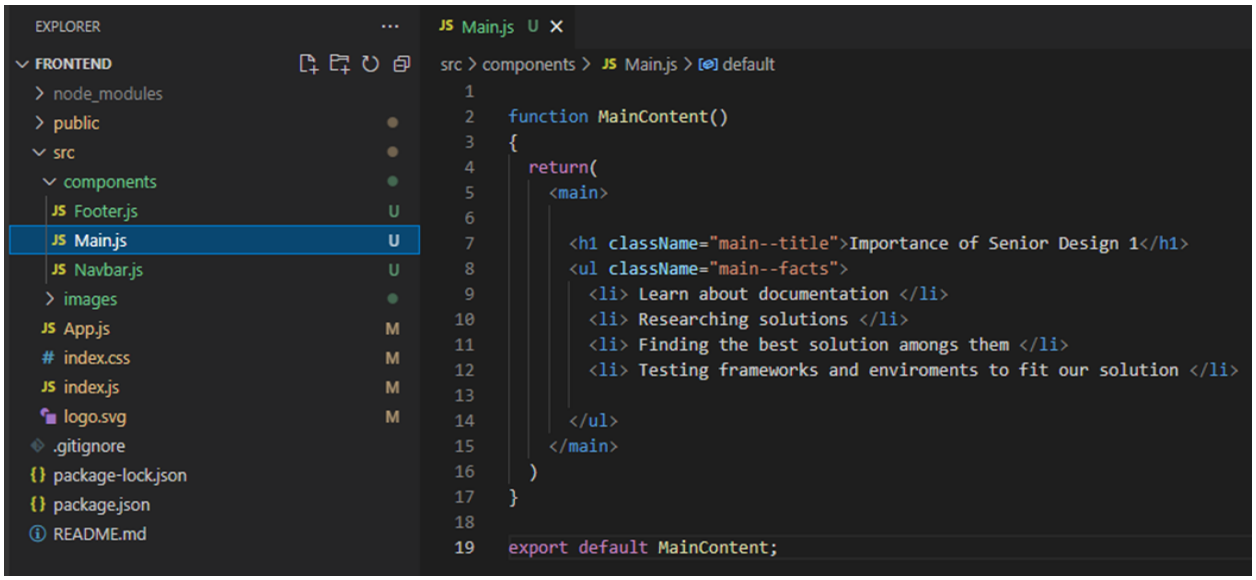


Figure 5.1.1.C. Main Page Component, Main.js

```
JS Navbar.js U
src > components > JS Navbar.js > Navbar
1   import React from 'react';
2   import logo from '../images/logo192.png'
3
4   function Navbar()
5   {
6     return (
7       <nav>
8         <img src={logo} className="nav--icon"/>
9         <h3 className="nav--logo_text">React Senior Design Example</h3>
10        <h4 className="nav--title">Navigaton bar component</h4>
11      </nav>
12    )
13  }
14
15  export default Navbar;
```

Figure 5.1.1.D. Navigation Bar Component, Navbar.js

```
JS Footer.js U
src > components > JS Footer.js > Footer
1   import React from 'react';
2
3   function Footer()
4   {
5     return (
6       <footer>Footer Component</footer>
7     )
8   }
9
10  export default Footer;
```

Figure 5.1.1.E. Footer Component, Footer.js

In Figure 5.1.1.C above is the component of an example main page. You can see on the left in the file explorer we have a components folder with Main.js, Navbar.js and Footer.js. In the Main.js we can see we implemented our component as a function and we are returning some JSX code which is just some header with a list of information. And at the end of our component, we need to export it with a name as they are in their

separate files and we want to import them to use our components in other files. And that is done with ‘export default MainContent’.

Figure 5.1.1.D and Figure 5.1.1.E shows the code for our navigation bar and footer components respectively. They follow similar style as they are JavaScript functions, they return JSX and we export them.

Since we have our components, now we can combine them in a single component and render them in React in Figure 5.1.1.F and 5.1.1.G.

JS App.js M ×

src > JS App.js > ...

```
1  import React from 'react';
2  import Navbar from './components/Navbar'
3  import Main from './components/Main'
4  import Footer from './components/Footer';
5
6  function App() {
7
8    return(
9      <div className="container">
10       <Navbar />
11       <Main />
12       <Footer/>
13     </div>
14   )
15 }
16
17 export default App;
18
```

Figure 5.1.1.F. Combination of Components, App.js

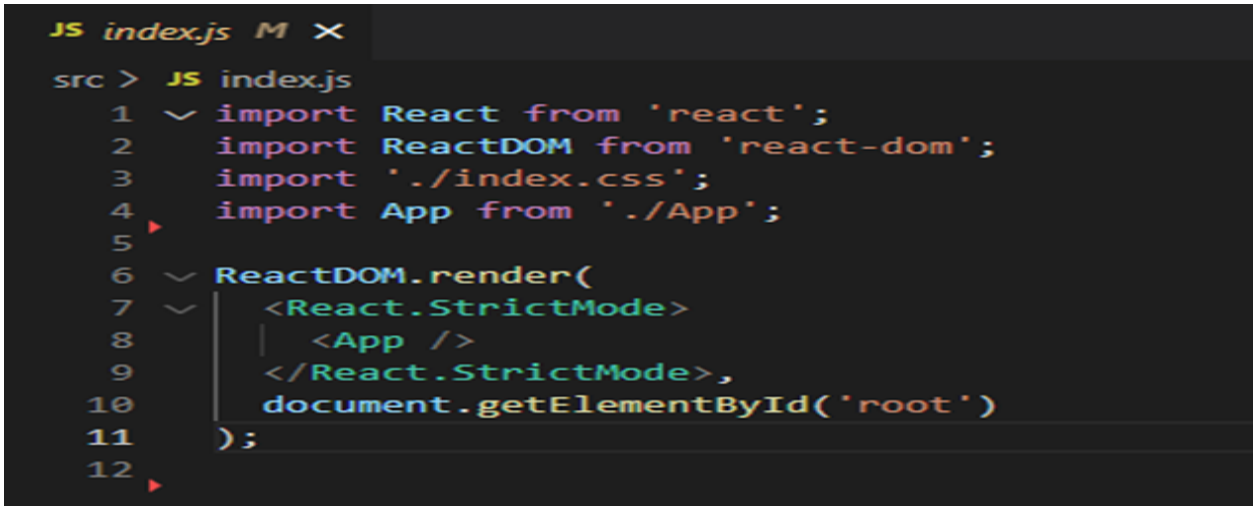


Figure 5.1.1.G. Rendering App.js

Figure 5.1.1.F is App.js, a component where we combine every component. We can see at the start we needed to import our other components we created from their respective folders. And here we can see how we use other components in other files such as <Navbar/>, <Main/>, <Footer/>. When we will render App.js, those components will be rendered in their places. And in this example we are rendering App.js in the Figure 5.1.1.G which is the code for index.js. We can see in index.js we imported react and react-dom which is needed for rendering, and App.js which has all of our components in one place. And then simply we render the app using ReactDOM. After that we can write ‘npm start’ while we are in the appropriate folder. Here is how it looks down after a bit of added CSS and logos of React in Figure 5.1.1.H.

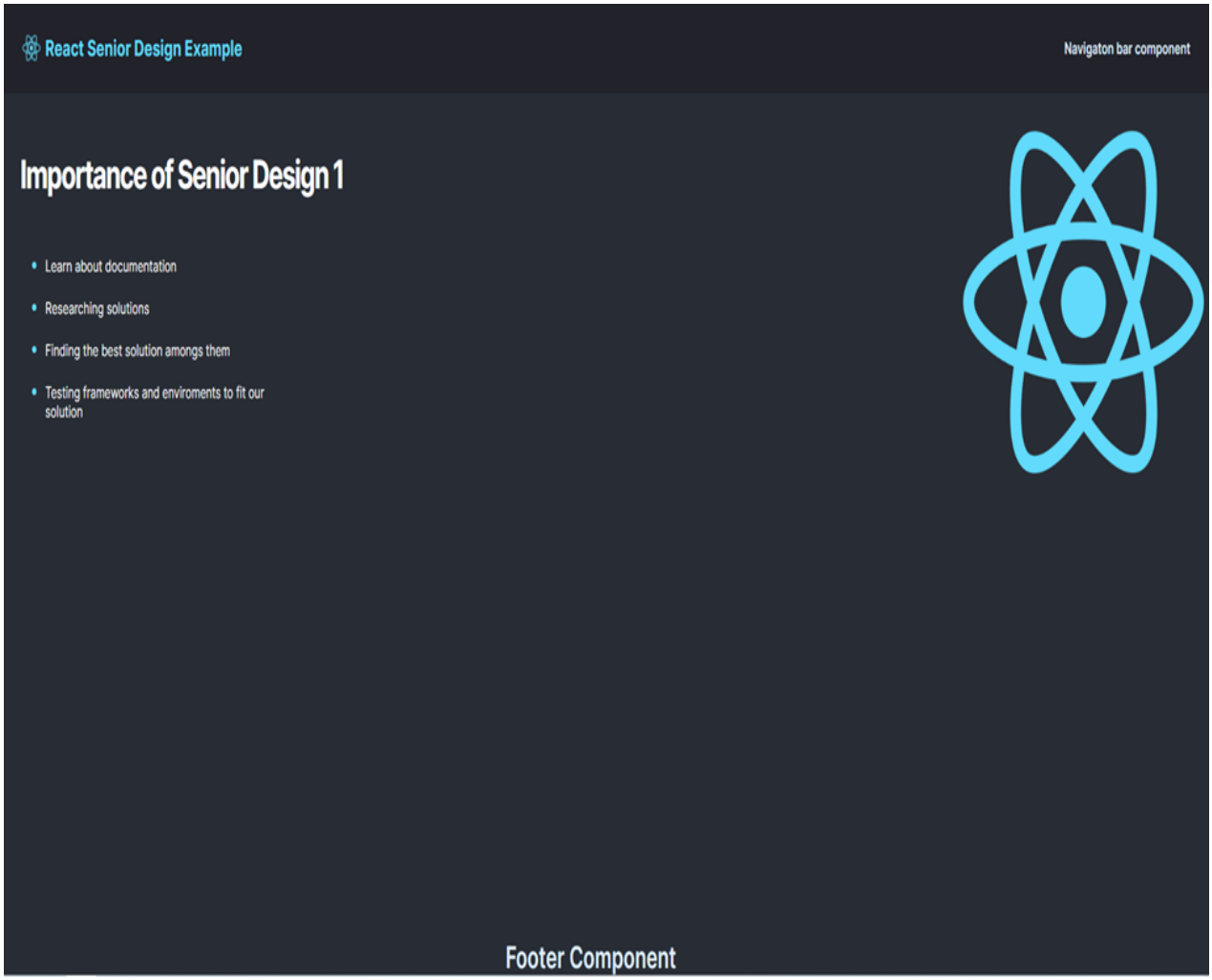


Figure 5.1.1.H. Sample Output for Code Written With Components

As we can see from Figure 5.1.1.H, our components are rendered in their place. Navigation bar up top, main page in the middle and footer is at the bottom.

For this example, we could say using components didn't give us any advantage, which is true. We could even say that it made things more complicated. We could just render everything in a single file and be done with it. This is true because it was a small example. When the projects get bigger and we have more parts to a webpage, it wouldn't be optimal to put them in a single file. That's when components come in handy and gives us an option to write a clean, more manageable code.

Props

Props in React are a way to pass in data to our components and make it so they can communicate with each other. The Props is just short for properties, and they are similar to HTML attributes. One of the rules of props is that they are not mutable. They are read only, therefore the component cannot change its props' data in any way. This ensures security and makes it so props only have uncompromised data when they are passed to a component.

Passing a prop to a component is very similar to passing values to a function. After we pass the prop, we can access its values and show those data however we want in our front end. We can also send data between components which is very good to make connections between components. Down below is an example of that in code using props in Figure 5.1.1.I.

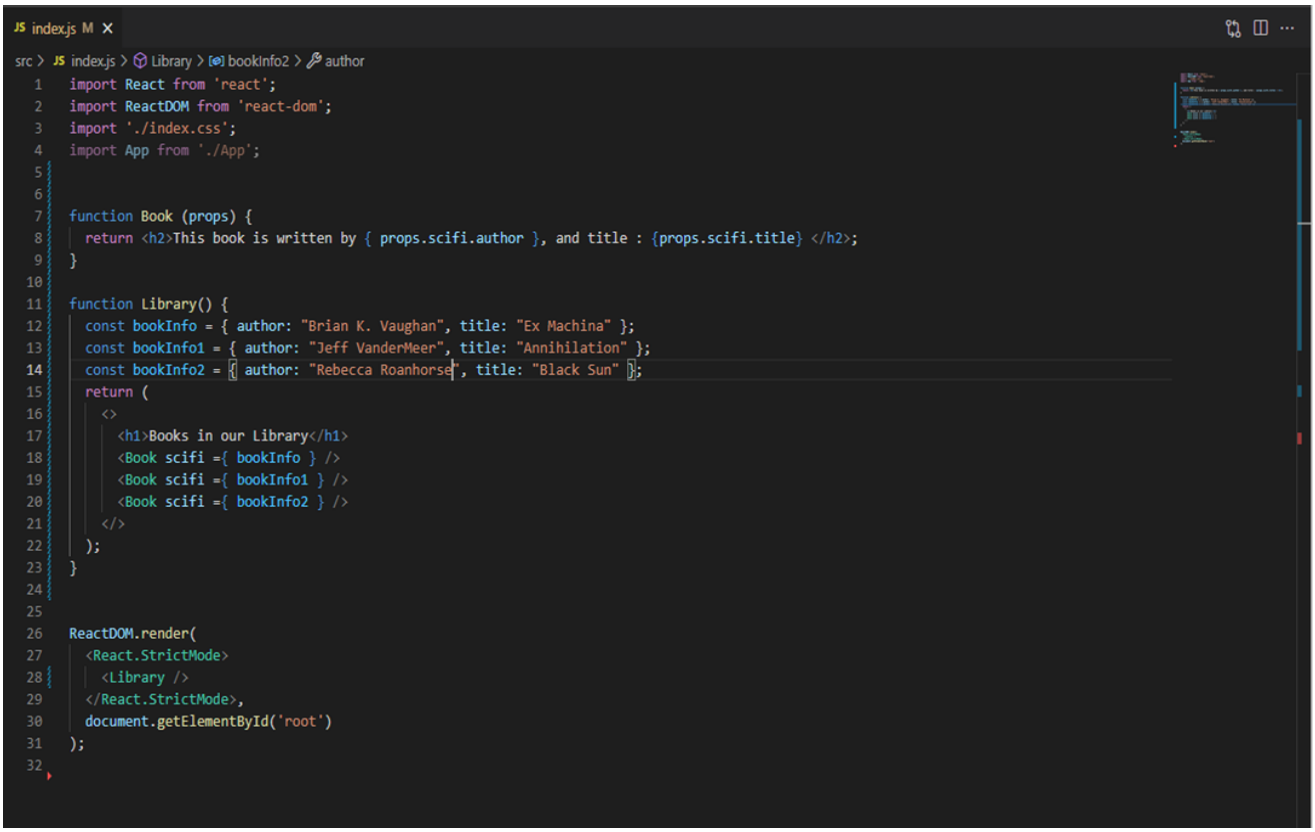


Figure 5.1.1.I. Props Example with Books.

In the code above, we have two functions, or we could say components. And as we can see, the Book function takes in props. That means we are allowed to use props in that function. And in the function, we are displaying some information wrapped in a header and we access {props.scifi.author} and {props.scifi.title}. That information, in this case props, was sent from the Library function.

In the Library function, we created three bookInfo objects with an author and title, and we are passing those bookInfo's to the Book function with the name 'scifi'. Then we use three Books in the library function. And in the end we just render the Library and it has the Books inside rendered as well as shown in Figure 5.1.1.J.

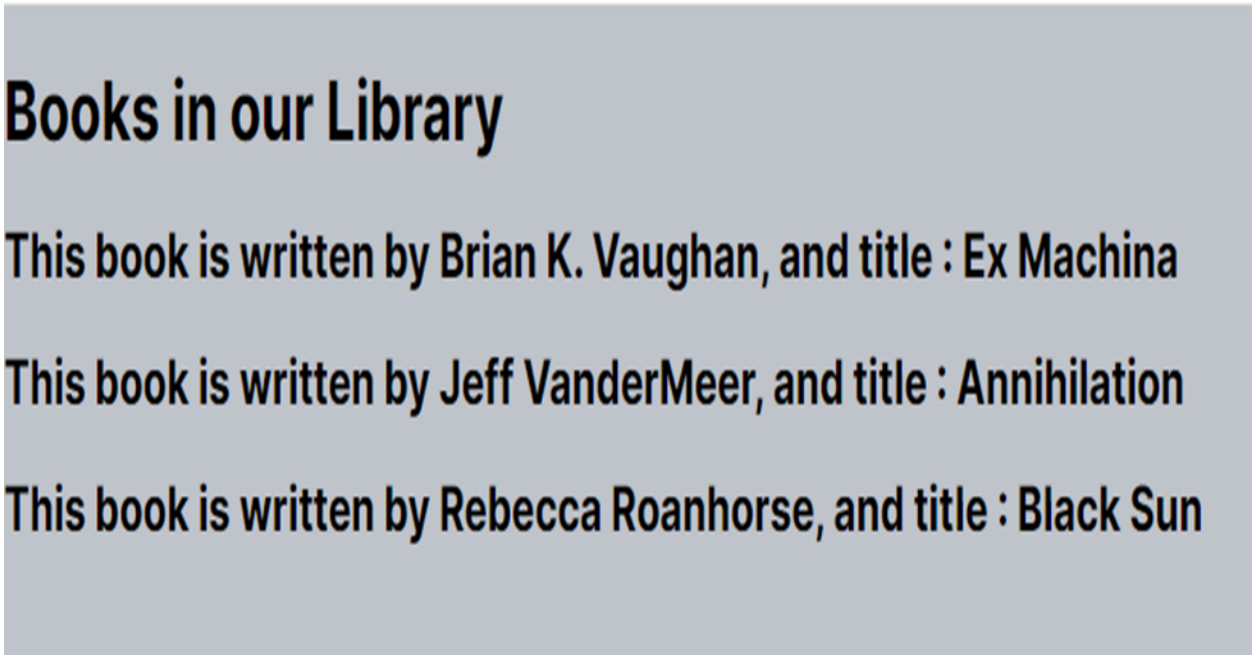


Figure 5.1.1.J. Output for Props Example

This is the output we get after we do 'npm start' in the terminal as seen in the Figure 5.1.1.J. What we essentially did was send three objects to the Book function from the Library function, and then we displayed those objects using props. This allowed us to save some space even though it is a very simple example.

Conclusion

Components and props in React essentially gives us the ability to write more effective code while shortening the amount of code we write. And also they are the concepts to move the data around between components dynamically. It is essential that we use props and components in this project to adapt writing cleaner, concise and effective code that gets the job done.

5.1.2 Advantages and Disadvantages of React

There are both advantages and disadvantages of using React and its libraries, down below we take a look at how advantageous React can be as well as the disadvantage that comes with it.

Advantages

- One of the main reasons why React comes to mind very often is that React is easier to learn than many other options and it's very popular. This popularity brings huge advantages to the people who want to get started with React. Tons of documentation, online tutorials and training options makes it very easy to get into React.
- React uses declarative programming instead of imperative programming. As we described above, the declarative paradigm is superior as it is intuitive, clean and overall easy to implement for user interfaces.
- React code uses components. Components are reusable pieces of code, and every component can be a piece of the UI. We can nest components and use them as a building block to our application. This also allows us to manage our code in a clean and efficient manner.
- React offers an incredible performance. As we mentioned above, react has its own Virtual DOM[7]. DOM is a programming interface for web documents. It represents the page. React renders only the components that actually change using the Virtual DOM and that is extremely efficient.

- React offers a JavaScript extension called JSX. It allows creation of React components even though it is not required to do so. JSX is the idea to mix HTML and JavaScript together[8]. It allows more readability to JavaScript code as JavaScript can be intimidating. But people with HTML experience can intuitively understand and develop with React using JSX.
- Addition to React, React Native is a framework to develop mobile apps using React library. Therefore, learning React can also open up possibilities to develop a mobile version of an app.

Disadvantages

- As we talked about the popularity and documentation, there is a slight disadvantage there. With constant updates and changes, it's not easy to catch up with React. And documentation coming with those changes and updates are not complete, which again makes React hard to keep up with.
- We talked about JSX in the advantages section. But even though JSX is there for simplification, it is still a learning curve. Some developers can see that as very unappealing and difficult.

Conclusion

React's advantages outweigh its disadvantages. Features like Virtual DOM, component-based programming, declarative programming and huge amounts of third party library options and support makes React a top choice on front end development. The toughest disadvantage of React is the learning curve. Starting from scratch with a lot of libraries and options can be overwhelming. And also, JSX can be intimidating as well. But all of our members in the group at least familiar with React and JSX are optional. Even if it wasn't optional I think we should spend time researching and learning it and turn JSX into our advantage.

5.1.3 React vs Other Solutions

React isn't the only option in the market for front end development. There are also options other than JavaScript. Like simple HTML, CSS and even PHP for pages. But since we have decided on using React, we are going to take a look at the other JavaScript libraries and frameworks. There are two frameworks that get compared to React with their popularity and features they offer when it comes to front end development with JavaScript. They are called Angular and Vue.

Angular

Angular is a full framework, unlike React being a library[9]. That comes with its advantages and disadvantages. Being a framework, Angular is consistent and clean. Angular can also handle routing and that makes it easier to switch between views. But compared to React, Angular has very limited flexibility with other libraries. Angular also has a very big learning curve as documentation and manuals are very lacking. Lastly, Angular's tons of built-in functionality can be seen as an advantage but having all that functionality from the start can be very overwhelming since with React we can just start out with basic libraries and expand our application as we see fit.

Vue

Vue.js is also a JavaScript library similar to React but not like Angular which is a framework. That means it is also lightweight as React. Vue and React are very similar and they are quite often compared to each other because they share the same kind of features. One of those features is that they both have Virtual DOM[10]. Both React and Vue have large community support as they are both very popular. But in the end React is still more popular, has more third-party library support and since we have been exposed to React, it is easier to learn. Another reason to choose React over Vue is because a lot of Vue's documentation is in Chinese and the language barrier is of course hard.

Conclusion

After comparing React to both Angular and Vue, we decided to use React as our front-end development library. Biggest factor for choosing React was the learning curve. We all have been exposed to React in our Processes of Object-Oriented Software development class and that is very valuable. Even though Vue and Angular offer similar features, React is just easier for us to adapt, learn and develop with its intuitive design and lightweight library. Therefore, we are using React for this project.

5.1.4 Most Common React Libraries

As we talked before, React offers a lot of third party component library options. These libraries are here to help us build better User Interfaces with less time. With these libraries, we don't have to create components like buttons, forms from scratch. And also we can customize these components to a point that it looks unique. That means we can make unique components easily and use those to build our projects. They are also crowd-sourced, which makes it easy to fix issues and develop features. Below are a few popular component libraries that React offers for us.

Material UI

Material UI which is now known as MUI is probably one of the most popular component libraries in the list as we can see it has a lot of stars in GitHub[11]. It has tons of components to choose, customize and use. Material UI is built with both JavaScript and TypeScript, and it's based on Google's Material Design. Material UI has extensive documentation and help online that makes it very beginner friendly. Spotify, Netflix, Amazon and a lot of famous companies use Material UI to build their User Interfaces. Here is a slider component from the Material UI website and the code for that in Figures 5.1.4.A and 5.1.4.B. We might take a look at these sliders as we need a slider in our map for this project.

Figure 5.1.4.A Slider Example.

```
<Slider
  getAriaLabel={() => 'Temperature range'}
  value={value}
  onChange={handleChange}
  valueLabelDisplay="auto"
  getAriaValueText={valuetext}
/>
```

Figure 5.1.4.B Slider Code.

Bootstrap

Bootstrap is the oldest library in this list which predates React itself. It is a HTML, CSS and JavaScript library that offers creation of nice looking, responsive web pages using layout components that we can put elements into. These components are composed of HTML, CSS declarations and JavaScript code in some occasions. React offers a library called React Bootstrap which is released in 2019[12]. With this library we can use Bootstrap functionality in React. All the components in this library are compatible with Bootstrap themes and on top of that we can customize and create our own themes. Here is a clean login form that we might use for our admin login in bootstrap with its code in Figure 5.1.4.C and Figure 5.1.4.D.

Figure 5.1.4.C. Simple Login Example.

```
<Form>
  <Form.Group className="mb-3" controlId="formBasicEmail">
    <Form.Label>Email address</Form.Label>
    <Form.Control type="email" placeholder="Enter email" />
    <Form.Text className="text-muted">
      We'll never share your email with anyone else.
    </Form.Text>
  </Form.Group>

  <Form.Group className="mb-3" controlId="formBasicPassword">
    <Form.Label>Password</Form.Label>
    <Form.Control type="password" placeholder="Password" />
  </Form.Group>
  <Form.Group className="mb-3" controlId="formBasicCheckbox">
    <Form.Check type="checkbox" label="Check me out" />
  </Form.Group>
  <Button variant="primary" type="submit">
    Submit
  </Button>
</Form>
```

Figure 5.1.4.D. Login Sample Code

Ant Design

Ant design brands itself as a design system for enterprise level products[13]. And they have 4 design values. Certainty, Meaningfulness, Growth, Naturalness. Ant Design offers more than 50 components. They also have Ant Design Pro, which is prebuilt templates and components that reminds Shopify. Sadly, Ant design doesn't offer as many theme options as Bootstrap or Material UI. And even though they have documentation, it's not as extensive as Bootstrap or Material UI again. Most famous site

that uses Ant Design for its User Interface is Binance. And that fits their enterprise level branding very much. We don't see our group using Ant Design but it's worth mentioning because it's still a popular library option for React and User Interface design. Down below is a colorful tagging component and a form component in Figures 5.1.4.E and 5.1.4.F. They both look very nice. We still might take those into consideration for our project since we need a tagging system.

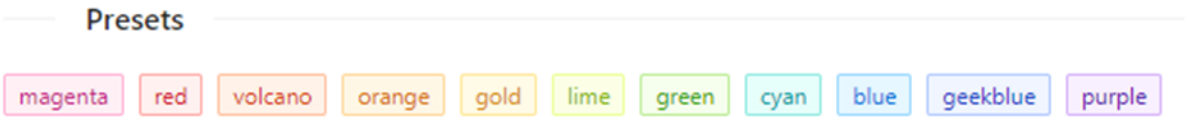


Figure 5.1.4.E. Tagging System Example.

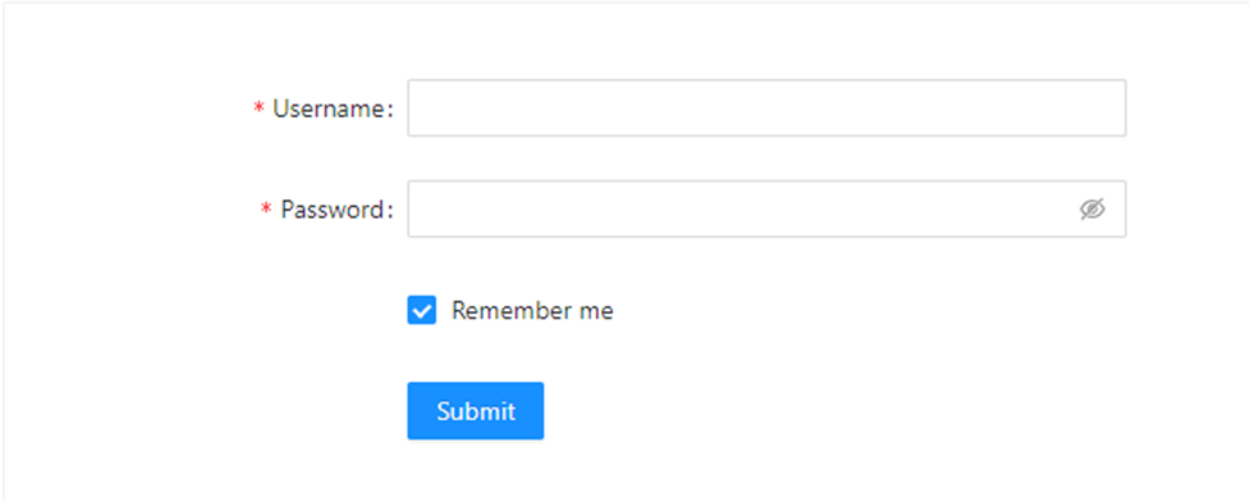


Figure 5.1.4.F. Another Login Sample.

Fluent UI

Last component library to consider in our opinion is Fluent UI. This library was known as the Fabric React. It is developed by the Microsoft development team[14]. Like other libraries we considered so far, Fluent UI also offers built in components. Since it's from Microsoft, they all have Microsoft Office feel to them. That might be a bad or good thing depending on the project. These components that Fluent UI offers, have compatibility with mobile as they support both iOS and Android devices. Down below is a toggle component that we might need to add for our map toggles in Figure 5.1.4.G. For example toggling between a heatmap or period map. It is very clean and basic which is what we want.

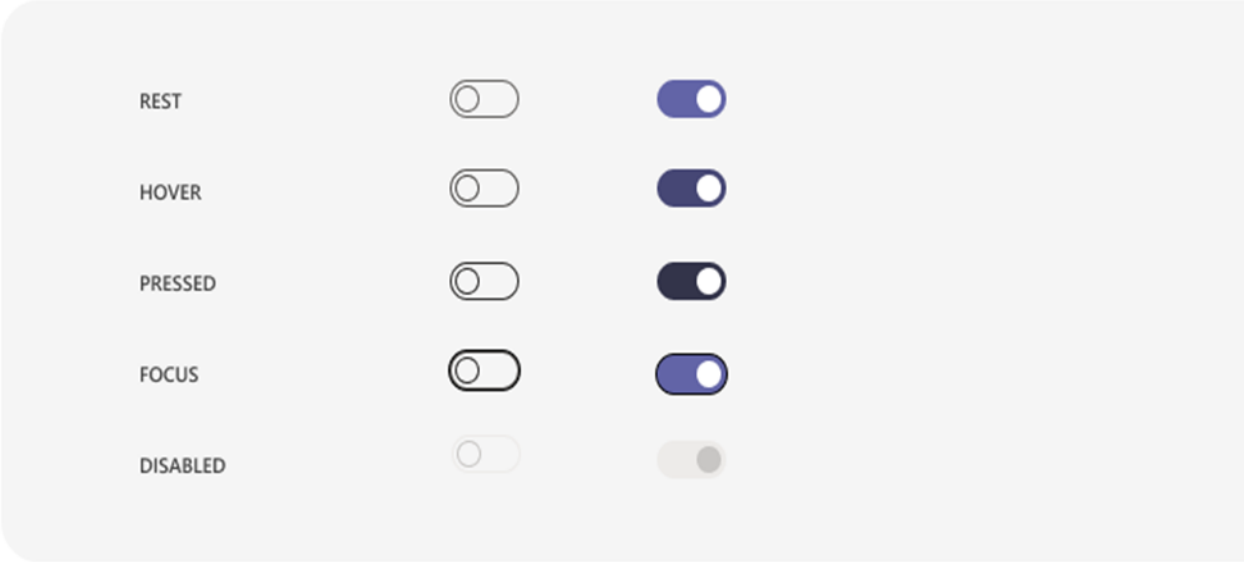


Figure 5.1.4.G. Toggling Example.

Routing

Addition to component libraries of React we want to use routing for navigation inside our project. Since we are using React, React router is the best option available with 2.4 Million GitHub users, more than 600 contributors and more than 3 Million downloads[15]. And also since it's used in the popular websites, and apps like Netflix and Discord there isn't a better other option to choose from. React router offers great documentation, examples, and tutorials so that it is beginner friendly. With nested

routes, we can show only the parts of the page we want. This is very useful. Below is an example from the React router website that shows in code how nested routes work in Figure 5.1.4.H.

```
<Route path="/" element={<App />}>
  <Route path="sales" element={<Sales />}>
    <Route path="invoices" element={<Invoices />}>
      <Route path=":invoice" element={<Invoice />} />
    </Route>
  </Route>
</Route>
```

Figure 5.1.4.H. Routing Sample Code.

As we can see from the example, the first path is the main page and it renders the App. Then every path has its own content and it's rendered when we enter that path. It is very clean and organized.

Conclusion

There are a lot of different kinds of libraries that React offers that we didn't mention yet. Those were the most common component libraries and there are more of them as well. This allows us to have a great variety of ways to implement this project. From those we mentioned up above, at this stage of the project, we are considering implementation using Bootstrap mostly. Bootstrap is familiar to us more than any other library so far. And having a lot of themes and responsive components, Bootstrap is a staple in the front-end development even though it is an older library. But in upcoming days, we might investigate additional libraries as we might into newer features and components that other libraries might have.

5.1.5 Fetching Data for Mapping

Fetching data to display in our front-end is a very important aspect of this project. We have enormous amounts of data to display with many categories. What we want is to have an easy way to fetch and show this data in the front-end. There are a few ways to accomplish this task. The two common ways to fetch data in React are called fetch() and Axios. They both allow us to make HTTP requests to the server from web browsers. They are both used widely but of course they have differences.

Fetch()

Fetch is a built-in option in React that does not need to be installed. One of the big differences between Axios and fetch is that fetch uses a two step process to deal with JSON data. First we make the initial request and that returns a promise and response object. Then we have to pass that response to the .json method to get our data[16]. Also in fetch we can handle the returned promises with then() and catch() methods as down below in Figure 5.1.5.A.

```
fetch(url)
    .then(response => {
        // handle the response
    })
    .catch(error => {
        // handle the error
    });
```

Figure 5.1.5.A Fetch() Error Handling.

Response also contains status codes that are very useful to us. With these status codes we can understand if our requests are successful, unauthorized, not found and so on. Down below are a couple of status codes and what they mean.

200 - OK, that means our request is successful

400 - Bad Request, as the name suggests that means our request cannot be processed.

401 – Unauthorized, this means we do not have access to that resource and we need to get authenticated first to get the response.

404 - Not Found, the famous 404 error means the server couldn't find the resource we have requested. Most common type of this error is that the URL is invalid.

Axios

Axios on the other hand is not a built in option like fetch() is. To use axios for our HTTP requests we need to install it like so using npm shown in Figure 5.1.5.B.

```
npm install axios
```

Figure 5.1.5.B Installing Axios.

After we install Axios, we can make requests. One big difference and advantage that Axios has is that data is automatically converted to JSON. Therefore we don't need to use the .json method for the data. Another advantage for Axios is that it supports almost all browsers, even the old ones like Internet Explorer 11. Error handling is very similar to fetch(), we can use the error codes in the response to either continue or handle depending on the code itself.

Conclusion

As we can see both Axios and fetch() accomplishes the same things while having differences and advantages over one another. To conclude Axios vs fetch() let's look at a POST request from both which is down below in Figures 5.1.5.C and 5.1.5.D.

```
// axios


const url = 'https://jsonplaceholder.typicode.com/posts'
const data = {
  a: 10,
  b: 20,
};
axios
  .post(url, data, {
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json;charset=UTF-8",
    },
  })
  .then(({data}) => {
    console.log(data);
});
```

Figure 5.1.5.C Sample Axios Code.

```
const url = "https://jsonplaceholder.typicode.com/todos";
const options = {
  method: "POST",
  headers: {
    Accept: "application/json",
    "Content-Type": "application/json;charset=UTF-8",
  },
  body: JSON.stringify({
    a: 10,
    b: 20,
  }),
};
fetch(url, options)
  .then((response) => response.json())
  .then((data) => {
    console.log(data);
  });
```

Figure 5.1.5.D. Sample fetch() Code.

Here we can see the differences we talked about in code. With an initial look without getting into the code we can see that the Axios code which is the first picture above is simpler and more concise than fetch() code. One difference pops up to the eye right away is that Axios uses data property to send the data compared to body property in fetch() which is a big syntax difference. Another important difference that makes the code simpler is how Axios converts the data to JSON automatically for us as we talked before. We can see response.json() part in the fetch() picture as a two step process needed to parse the data.

In the end, we believe Axios is a better option to use to make HTTP requests than fetch(). Even though fetch() can accomplish the same things, Axios' simplicity and added functionality as well as more compatibility options makes it a better option for this project.

5.1.6 Leaflet

Here we will delve into how we will go about visualizing the letters on the map and the services we will employ to get the effects the sponsor requires.

What is Leaflet?

Leaflet[17] is a well-known and used open-source JavaScript library for mobile-friendly interactive maps that supports both mobile and desktop platforms developed by Vladimir Agafonkin. Leaflet natively supports Web Map Service layers, GeoJSON layers, Vector layers, and Tile layers with many other types available through plugins. It loads data from map files and gives the developer tools to style the layout and create components that interact with the map data such as creating and selecting markers with detailed information on click. Beyond everything else, Leaflet has laser focus on “simplicity, performance, and usability”, which can be quickly seen in Figure 5.1.6.A and Figure 5.1.6.B as they display a simple, quick, and yet informative map implementation.

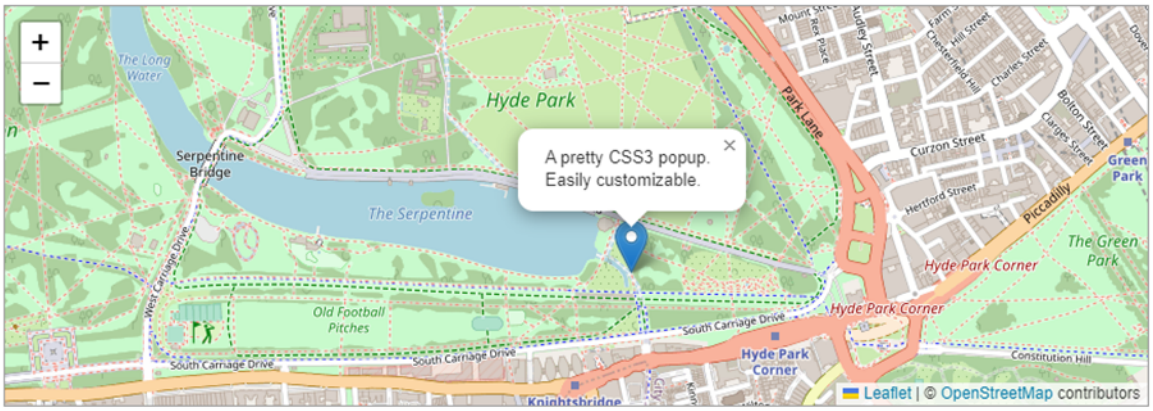


Figure 5.1.6.A. Simply Leaflet map Implementation

```
var map = L.map('map').setView([51.505, -0.09], 13);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);

L.marker([51.5, -0.09]).addTo(map)
    .bindPopup('A pretty CSS3 popup.<br> Easily customizable.')
    .openPopup();
```

Figure 5.1.6.B. Simply Leaflet map Implementation code

5.1.7 Benefits of Leaflet

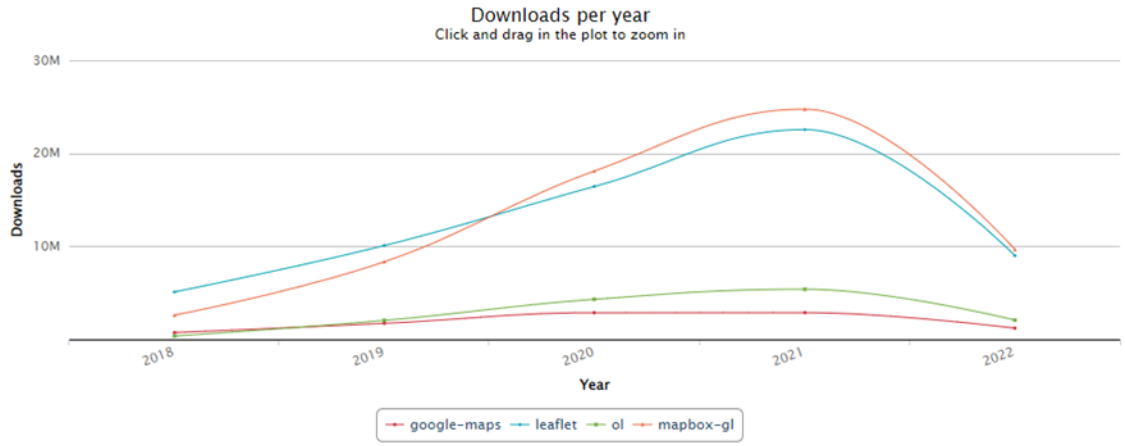
Leaflet provides many important features that will enable us to fulfill both the project requirements and potential stretch goals.

- Leaflet is a free library. While the sponsor is fine with small, one-time payments, many competitive services require time or usage-based payments for features we would not even use, which is not something we want.
- Leaflet is highly flexible and customizable. Flexible in its small size and customizable with its vast support. It comes with a plethora of options for design and layer customization and is open for additional features for viewing data on the map.
- Natively functions with “slippy” maps. Modern maps which let a user zoom and pan around. This is important for our implementation as a user will need to crop into specific sections of a map to view detailed information regarding the letters.
- Leaflet is simple to implement and use. As this is a one-off project, we need a library or service that can quickly be learned for use for the quickest turnaround.
- Allows for multiple, customizable Layers. Per the requirements, we need a way to display data that will create a layer over the base map, and more the base map to change based on the user’s current research. Like the base layer changing with a time slider and overlays showing historical maps.
- Lightweight framework with plenty of plugin options for extended versatility to accomplish goals such as the heatmap. A primary requirement that allows the implementation of several map effects such as heatmaps and mini charts.

- A large community of developers who use Leaflet. If we have issues with our development, there are a host of internet resources that could provide potential solutions or solutions to similar problems. Additionally, as part of our sponsor group, we have access to someone highly familiar with Leaflet that has offered to provide any help and guidance we might need.
- Allows creation of Coordinate Reference System (CRS). Requirement for plotting locations of Keats letters to reference location density and movement over time.
- Supports ReactJS development. A primary requirement as our development is planned around ReachJS.
- Mobile friendly. While not a strict requirement, would be advantageous for either our stretch goal or for future development making this project fully mobile supported.
- GeoJSON supported. Since fitting historical maps is a requirement, most geo-rectification programs output the file in the GeoJSON format.

5.1.8 Leaflet vs. Alternatives

There are many suitable and popular alternatives to Leaflet that are downloaded and used every day as shown in Figure 5.1.8.A. Each service has its own use cases with pros and cons. Leaflet stood out among them but seeing competition allowed us to figure out what is and is not important for our specific use case.



Total number of downloads between *2018-04-22* and *2022-04-22*:

package	downloads
mapbox-gl	63,433,430
leaflet	63,147,571
ol	14,143,637
google-maps	9,362,999

Figure 5.1.8.A. Popular map based data visualization packages compared. All data comes directly from npm and charts powered by highcharts JS which is provided under a CC BU-NC 3.0 license.

Leaflet vs OpenLayers

While both are popular JavaScript libraries used for web-app based interactive mapping, they differ in their immediate uses. Leaflet is relatively small and simple to implement compared to the high-performance, feature packed OpenLayers[18][19] library. The simplicity allows us to learn and implement our project solution much faster. While the small size heralded for optimal use on mobile devices initially seemed detrimental for our needed intricate uses, its vast community has provided plugins for every requirement we might need. This allows us to keep the library small while only bringing in necessary components.

Leaflet vs Mapbox

Mapbox[20][21] is a complex and feature-rich mapping library that supports most, if not all, requirements for mapping Keats. Where it falls short is in two main areas: Learning curve and cost. The most prohibitive feature of Mapbox for us is the cost. The pricing for it is having a cost per monthly user, which does not fit with the sponsor's vision.

Leaflet vs Google Maps

Google Maps[22][23] is a free open-source mapping service, not just a library, for JavaScript that is widely used (an understatement). It also offers complex and powerful features such as traffic and transit data unlike Leaflet. This is not something we need, however. While Google does charge for Maps, they only do so at high levels of usage, which would be far more than our usage at this, and future, stage. The main limitation is that it forces use of the default Google base layer and we need to be able to impose layers with historical maps.

Leaflet Additions

Important plugins for features such as time slider, heatmap, minicharts, etc.

For Leaflet to have all the functionality that we need, its supported plugins will be used.

1. A required feature is for data to be time dependent and the user able to alter the data that shows up with a slider. While Leaflet has support for specific use case time sliders, the general timelineSlider creates a customizable timeline slider with user designed functionality.

2. Another required feature is displaying data within a heatmap. There are several options from incredibly in-depth add-ons that allow detailed user decisions to simplistic ones that have predetermined settings to show concentrated data points.

3. Many addons would let a user interact with a personalized layer to give them freedom to make notes directly on the map. This feature would be a small stretch-goal not directly linked with the project.

4. Leaflet.Search allows for searching markers and features over multiple layers with autocomplete. While Leaflet natively supports this functionality, this plugin simply makes the implementation quicker and easier.

I don't think it'll be necessary in the future, but there is also a way to integrate Google Maps in Leaflet.

5.1.9 Map Warper

What is Map Warper?

Map Warper[24] is a free open-source tool used for geo-rectification developed by Tim Waters. It has a simple interface and involves dropping several anchor points into historical and contemporary maps. An example of this process is shown in Figure 5.1.9.A where 18 control points are selected for the modern and antique maps. Figure 5.1.9.B shows the geo-rectified map overlaid on the modern map. The internet has several resources and examples of use. This tool is necessary as the historical maps we need to use are not completely accurate compared to today's maps using our modern technology. To make them suitable overlays, geo-rectification can interpolate between control points of the historical map and warp it to accuracy as best as possible. This is an imperfect technique, but the sponsor understands that process and is fine with it.

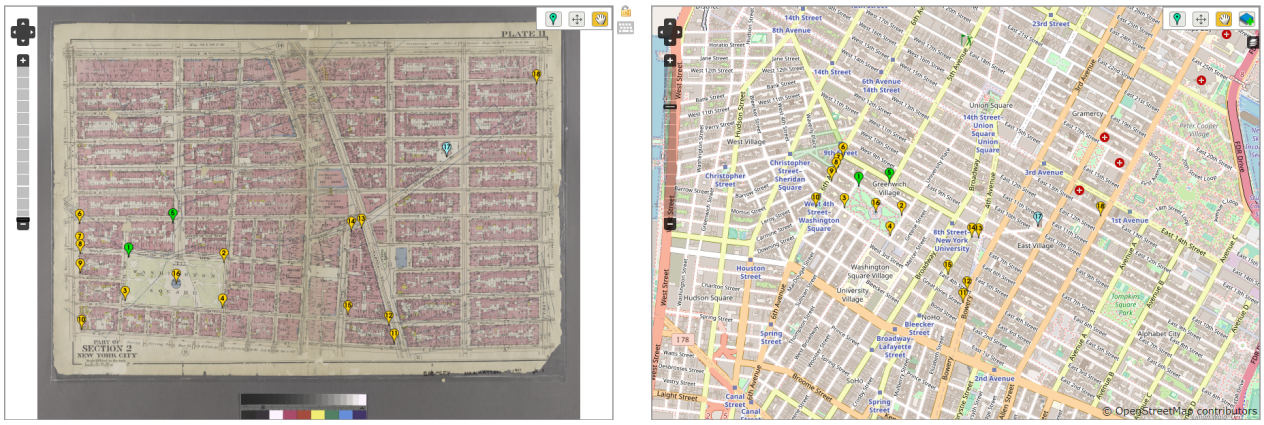


Figure 5.1.9.A Control Point Placement

Figure 5.1.9.B . Georectified Map Over Modern Map

5.2 Back-End System

5.2.1 Node.js

NodeJS is a back-end JS runtime environment that is used to execute JS code and it will be the primary utility we use to develop our API. NodeJS can be described as an event based architecture that utilizes asynchronous I/O, meaning processing can occur before transmission is complete. NodeJS is built on Google's V8 engine, which serves as an excellent foundation for the framework. Node, in conjunction with Express and Sequelize, will be used to handle our API[25].

Benefits of Node

NodeJS is among the most popular frameworks used by developers and tech start ups and more importantly it offers many significant advantages that aid in meeting the requirements for the application[26].

- Speed and efficiency
    - Library includes a significant amount of utility as it notably decreases the amount of code written.
    - Moreover, the tools in the library also allows for fast code execution aided in part by its asynchronous design.
- Scalability
    - Once again thanks to its non-sequential design, NodeJS is well equipped to handle a large number of simultaneous events and processes.
        - It should be noted that this aids a potential long term scope for this project beyond administrative uses: expanding to general users.

- Development process
    - Simple language based in regular JavaScript, reducing the learning curve for developers.
    - Furthermore, certain JavaScript can be shared between front-end and back-end, leading to less files and code being written to produce the end product.
    - Simple deployment is immensely beneficial towards delivery and even unit/component testing. Also beneficial when producing an MVP (minimum viable product).
- NPM (Node Package Manager)
    - Extremely influential development tool for NodeJS that includes an immense amount of libraries and development utilities such as templates and modules.

Drawbacks of NodeJS

The strengths of NodeJS are met with some varying drawbacks. While some of these could be significant, fortunately they are mostly either do not deter the meeting of requirements or can be addressed by the supplementary use of Express, which will be further explored later. It is still important that they be addressed however, when considering long term scope and scalability of the application.

- Susceptible to bottlenecking with heavy computation
    - Fortunately this should not present any major issues within the finite scope of this application as it currently exists. However, depending on the potential long term deployment of the application to a wider audience it would serve and accordingly the more extensive functionality that could be added, potential bottlenecking would need to be considered.

- Natural drawbacks of being open sourced
    - As NodeJS is open sourced, many of its tools in NPM can be underdeveloped or under-documented, which can inevitably lead to poor structure and quality.
        - As the functionality of the application is fairly straightforward and static, there should not be any real need for niche tools that cannot be solved by well documented mature modules. However once again, this may not be the case when considering long term scope.
- Lacking in some web development component utility
    - Does not include immediate routing and controller tools.
        - Compensated when supplemented with Express.

Node Dependencies

- Tools/Utilities
    - npm
        - Package manager utilized by NodeJS
    - Gyp
        - Allows for the compiling of native, add-on modules used by Node
    - Gtest
        - (Google Test) Unit testing library
- Libraries
    - V8
        - JavaScript engine used in Google Chrome that is the foundation of Node

- Zlib
    - Utility library for compression and decompression (zipping and unzipping), implements Gzip.
- Libuv
    - Supports asynchronous I/O functionality dependant on event loops
- OpenSSL
    - Package that gives you access to all OpenSSL commands which are used for security through key generation, certificate installation, and certificate authentication.

5.2.2 Express: Overview

Express and Node will be used in tandem on the back end to support our backend functionality. These two frameworks for javascript are commonly used together, and it is especially effective in configuration and customization of routes in an application[27]. Express, much like Node, is also JavaScript based, allowing developers to code in fewer languages between frontend and backend development. As it is an extension of Node, it also shares many of the same strengths and weaknesses.

Express JS would be a great fit for this project because it is a web framework built to work with Node.js. It provides the programmer with certain tools and functions that are very useful when writing a web application. These tools can be used to write better, safer code, especially when creating routes for the website. Routing is an important feature for any website, and programmers must make sure that they are implementing this in a way that does not expose any security vulnerabilities. Since routing is built into Express, we do not have to do all the legwork to manually route the client to different pages on the server. This saves us lots of time and effort on the backend, allowing for that time to be spent on other necessary features like API routes and unit testing. Another feature Express provides us with is that we can listen for a certain port on the server, and connect to it using Node and Express. Express is a middleware, meaning

that we can use it to make decisions to give appropriate responses to client side requests. [28]

We have also chosen to use Express since it is written in javascript, a language that we are going to be using on both the front and back end of our application, with Node.js and React.js. This allows the majority of our project to be written in javascript with the use of these frameworks, so all group members will be able to view different parts of the project and have the ability to read and learn how each piece is implemented. Most of the members in our group are already familiar with javascript to some extent, so everyone will be able to somewhat understand the code. We feel that Express will allow us to build a website that is scalable and efficient, giving our users the best experience possible on a complex website.

5.2.3 Framework Alternatives Considered

Prior to settling on a single framework, several frameworks were considered to deduce which would best suit both the needs of the sponsors and their requirements as well as the developers.

Elixir

Elixir is a framework sharing many similarities with Node, especially in its strengths. As a result, it was one of the candidates we considered when deciding on a framework appropriate for our project.

Strengths:

- Asynchronous
    - Much like NodeJS, Elixir can handle requests and posts simultaneously in real time without slowing down the application. This leads to a much faster, smoother user experience[29].

- Scalability
    - Can utilize multiple virtual servers that allow applications to run on multiple nodes that communicate with each other, allowing the application to run the application more efficiently.
- Fail safe
    - Elixir has some built-in mechanisms that allow for a margin of error in runtime, allowing the application to function even during certain scenarios where it would fail, as servers can communicate process failure to its dependent processes so the problem can be fixed.

Despite these apparent strengths of Elixir, it has a few drawbacks that can make it difficult or less efficient to implement in the context of our project.

Drawbacks:

- Less developed
    - Even though it was first released in 2012, Elixir only recently became increasingly popular and as a result has a much less developed infrastructure than Node. Concurrently there are also less experienced developers with the technology, so there are less resources to reference in the case of errors or bug fixing. Node is far more developed in this regard, and there are far more tools available to developers to help improve their development process.
- Processing
    - Elixir is a fairly inefficient framework in terms of processing speeds, as it is inefficient with raw CPU power. Large amounts of intensive calculations can make it difficult to run certain processes asynchronously. While this isn't inherently avoided by selecting Node, it is another drawback that pushed us away from selecting Elixir.

- Learning curve
    - Along with being less developed, Elixir would present a far steeper learning curve as it would require our developers to learn a new technology with less resources for learning that technology. This is not an issue with Node however, which has a far wider breadth of educational resources for learning the tool.

As compelling as Elixir was as an option for our framework, it was ultimately not considered viable enough to learn as it was outweighed by its far more suitable alternative.

PERL

PERL is another high level programming language that is a fairly popular development tool.

Strengths:

- Flexible
    - Supports function programming languages, procedural, and objects, allowing it to be very versatile and powerful[30].
- Secure
    - High complexity programming language, better suited to handle data encryption.
- Mature
    - Fairly well developed and open source, allowing for a wide variety of support from developers and no shortage or helpful resources.
- Text processing

    - Can process and handle text very effectively using its text-processing facilities. Text files can be manipulated and formatted easily using PERL.

While these strengths are noteworthy, they are overshadowed by the drawbacks in the contexts of this project.

Drawbacks:

- Not specialized
    - Not created for web development and a large user base, compromising its potential for scalability.
- Development
    - Not a widely popular language, preventing it from having a lot of backing and support for developers looking for bug fixes or development issues.

As one of our larger stretch goals focuses on scalability and expanding our application to a wider audience, as well as the lack of developer support, PERL was determined to not be best suited for our project.

ASP.NET

The last and one of the closest contenders for the primary framework was ASP.NET. Functionally, it can aid us outside of just back-end services and for this reason it was highly considered.

Strengths:

- MVC
    - ASP.NET utilizes an MVC (Model, View, and Controller) architecture that allows developers to accurately control input, processing, and outputs in an application[31]. Each part works in tandem with one another, which can greatly reduce coding time and testing. This was a major factor of consideration as our application was initially planned to have a straightforward layout. This model ultimately leads to enhanced scalability, ease of coding, and improved performance.
- Offers useful tools
    - Includes just-in-time compilation, which compiles code during execution, native optimization, and caching services.
- Structure
    - ASP.NET's model can allow tasks to be performed easily with less overhead. This would have been helpful as an option as our most complicated functionalities may have been streamlined, however our project as whole would not have been greatly improved or made by this.
- Modular
    - Parly as an extension on the point about its architecture, the design of ASP.NET allows developers to easily manage their systems as configurations are implemented as plain texts, so settings can be manipulated and deployed easier without any excessive server restarts. This would be especially helpful during test processes.

ASP.NETs simplicity and strengths were highly influential and compelling when we were in the process of selecting frameworks. While the listed strengths were certainly considerable, we found that they were all either negligible in the context of our project, better implemented by NodeJS, or simply outweighed by some of the weaknesses of ASP.NET.

Drawbacks:

- Limited resources
    - The documentation for ASP.NET is very lackluster and would thus set us back, especially in consideration of the following point.
- Lack of experience
    - An extension on our previous point, many of our colleagues as well as sponsors weren't greatly experienced with ASP.NET and that fact paired with the lack of sufficient documentation would add constraints towards meeting deadlines and the development process as a whole.
- Expenses
    - Possibly the largest factor that turned us away from ASP.NET were the potential expenses. Relative to the previous point, large scale development is hindered financially seeking developers experienced in specific technologies. More relative to our circumstances, there would be potential licensing costs to consider, for example in using Visual Studios, that might exceed our budget in the long term span of our application, such as expansion into commercial use.
- Flexibility
    - The rigidness of ASP.NET's MVC structure would conflict with the interests and requirements of our sponsor as well, as complications might arise when working with the CHDR server and related data.

5.2.4 Implementation and Integration

Node Server Application

NodeJS is written in primarily JavaScript syntax and can import premade modules in the Node library. One of the primary modules is the http module, an essential module for transferring data over HyperText Transfer Protocol. This is utilized in creating HTTP servers that can listen to server ports and provide responses back for the client side. Such servers can be initialized using the createServer method in the http module. Any function passed into the createServer method will then be executed whenever anyone tries to access the computer on the port that is specified under the listen method, as exemplified in Figure 5.2.4.A.

```
var http = require('http'); //http module

//create server object
http.createServer(function (req, res) {
  res.write('output'); //write response sent to client
  res.end(); //end response
}).listen(8080);
```

Figure 5.2.4.A. http module server initialization

Something of note is the *req* or the request argument, which is passed in as the Query String. The query string can be parsed and split and can aid in performing specific responses based on user request.

Modules in general can be initialized as variables and accessed accordingly. This is the primary mechanism utilized in assignment and routing in the application, as depicted in Figure 5.2.4.A. A point of note should be the simplicity in implementation, as Express can utilize express libraries to very easily define index and user routing as depicted as well as include line efficient error handling. Ultimately processes such as routing, rendering, and error handling become much more streamlined and efficient while functioning smoothly, especially for a project of our scale.

JS app.js > ...

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Figure 5.2.4.B. Sample app.js

Other Notable Functions/Utilities

Node and Express include a variety of other notable functionalities that allow for a smooth development process and an effective application.

- File I/O(File System): fs.readFile() and its counterpart, fs.appendFile() can be used to both read in file content and create a new file, respectively. The latter method will append specified content to the end of the file specified if it is not an empty file. Other methods such as fs.open() can be used similarly to open a file flagged “w” for writable or otherwise create an empty file whereas fs.writeFile() can be used to replace a specified file and its contents if such a file exists, otherwise creating a new file with the specified content.

```
var http = require('http');
var fs = require('fs');

http.createServer(function (req, res) {
  fs.readFile('sample.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);

fs.appendFile('file1.txt', 'content here', function (err) {
  if (err) throw err;
  console.log('Saved!');
});

fs.open('file2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});

fs.writeFile('file3.txt', 'content here', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Figure 5.2.4.C. Servers functions in Node

- File Handling(File System): Files can be deleted using the fs.unlink() method and files can be renamed using fs.rename().

```
var http = require('http');
var fs = require('fs');

fs.unlink('file2.txt', function (err) {
  if (err) throw err;
  console.log('file removed');
});
```

Figure 5.2.4.D. File removal function

- URL: The URL module is extremely effective at taking in a web address and parsing it into readable components that can be used for further functionality, as exemplified in Figure 5.2.4.E.

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2022&month=march';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2022&month=march'

var qdata = q.query; //returns an object: { year: 2022, month: 'march' }
console.log(qdata.month); //returns 'march'
```

Figure 5.2.4.E. URL Formatting.

- Returning content to users in the form of file content can be accomplished with a combination of the previous functions by initializing the respective files as shown in Figure 5.2.4.F.

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Figure 5.2.4.F. Returning content

Events in Node

Events in Node encompass most actions happening on the computer to interact with the application. One way to handle events is utilizing the EventEmitter objects.

Emitters can be used to fire events as part of responses as shown in the example in 5.2.4.G.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

//Initialize event handler:
var myEventHandler = function () {
  console.log('event handler created');
}

//Assign event handler:
eventEmitter.on('output', myEventHandler);

//Fire the output event:
eventEmitter.emit('output');
```

Figure 5.2.4.G. EventEmitter

Querying

Databases can be queried very efficiently using Node as shown in Figure 5.2.4.H using a sample MySQL database to easily read and write to a database.

```
con.connect(function(err) {
    if (err) throw err;
    console.log("connected");
    con.query(sql, function (err, result) {
      if (err) throw err;
      console.log("result: " + result);
    });
  });
```

Figure 5.2.4.H. Querying Example

5.2.5 Object-Relational Mapping

The primary goal of an Object Relational Mapping (ORM) is to map and convert data in relational databases into objects in the respective programming language[32]. This serves to map details between two different data sources that cannot normally operate together. This will be convenient for this application as we are working between different databases and servers. ORMs will cut development time as well as it prevents developers from needing to write raw SQL queries as well as allowing for a much more convenient development environment.

5.2.6 Sequelize

Sequelize is a promise based Object-Relational Mapping (ORM) utility for Node which effectively serves the purpose of fulfilling promises between NodeJS requests and databases, acting as an intermediary buffer[33]. Promises are a fairly unique event in Node that will be guaranteed to produce a result in the future, which can be either fulfilled or rejected, which is a successful response and a failure, respectively.

Sequelize Implementation

Authentication - To utilize Sequelize, a sequelize instance will be required to allow a URI connection to be passed. Connections can be tested as shown in figure 5.2.6.A using sample databases.

```
const { Sequelize } = require('sequelize');

// pass sample URI
const sequelize = new Sequelize('sqlite::memory:') // sqlite

try {
  await sequelize.authenticate();
  console.log('connected');
} catch (error) {
  console.error('failed conection', error);
}
```

Figure 5.2.6.A. Authenticate Function

Modeling - Models in sequelize effectively serve as representing a table in the database in the form of a model instance associated with an entity in the database. Figure 5.2.6.B shows a sample database represented using modeling in sequelize on a sqlite database.

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', {
  // define attributes
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING
  }
}, {
  // more models params
});
```

Figure 5.2.6.B. Modeling in Sequelize

Using this example, we can exemplify the usage of instances in Sequelize, which will allow us to appropriately send and receive data to the database. We can assume a set up as shown in Figure 5.2.6.C.

```
const User = sequelize.define("user", {
    name: DataTypes.TEXT,
    inputValue: {
      type: DataTypes.TEXT,
      defaultValue: 'default'
    },
    age: DataTypes.INTEGER,
    cash: DataTypes.INTEGER
  });
```

Figure 5.2.6.C. Instance model

Creating instances - Instances can be initialized as shown below. Note that despite instances being a class, they are constructed using “build” as opposed to “new”. In Figure 5.2.6.D, the entry has been initialized but not stored in the database asynchronously, which can be accomplished using the await save() method.

```
1   const name = User.build({ name: "Name" });
2   console.log(this.name instanceof User); //:true
3   console.log(this.name.name); //:"Name"
```

Figure 5.2.6.D. Initializing Instance

Updating instances - Another relevant function is updating values in instances. This is easily accomplished through basic reassignment of the properties of the object. Note that this will not automatically update in the database and the save method will need to be called again for the changes to be saved, as is shown in Figure 5.2.6.E.

```
1   const newUser = await User.create({name: "User1"});
2   newUser.name = "User1";
3   await newUser.save(); //update value in database
```

Figure 5.2.6.E. Updating instance values

A noteworthy compression of this code that will allow us to change multiple properties at once is the use of the set method as depicted in Figure 5.2.6.F.

```
const newUser = await User.create({name: "User1"});
newUser.set({
    name: "User2",
    adminLevel: "admin"
});
await newUser.save(); //update value in database
```

Figure 5.2.6.F. Updating instance values using set

Deleting instances - Instances can be removed extremely easily by simply calling the destroy() method on the instance as shown in Figure 5.2.6.G.

```
const newUser = await User.create({name: "User1"});
await newUser.destroy(); //deletes instance
```

Figure 5.2.6.G. Removing instance

Queries: *Where Clause* - The where clause is the primary tool used in filtering POST queries. It simply requires the parameter to filter by and the values to filter for, as depicted in 5.2.6.H.

```
Post.findAll({
    where: {
      adminLevel: "admin"
    }
  });//search for all admins
```

Figure 5.2.6.H. Search for users with admin levels

Application Code

By utilizing Express and Sequelize, we were able to properly implement create, update, and delete functions, as well as basic login, for admin users in the application. The following libraries in Figure 5.2.6.I were implemented in our routes.

```
const router = require("express").Router();
const db = require("../models");
const bcrypt = require("bcryptjs");
const dotenv = require("dotenv");
const { Op } = require("sequelize");
```

Figure 5.2.6.I. Libraries

The login function is shown in Figure 5.2.6.J. Some elements of note include the usage of bcrypt to hash the password. Users are searched by the email and password parameters passed in the request. Invalid users will successfully receive a 401 error, otherwise a successful login will return status 200.

```
router.post("/login", async (req, res) => {
    const {email, password} = req.body;
    let hashedPassword = bcrypt.hashSync(password, 10);
    try {
        // find user
        const User = await db.user.findOne({
            where: { email: email, password: hashedPassword },
        });

        if (!User)
            return res.status(401).json({ id: "", token: "", error: "Invalid Credentials" });

        return res.status(200).json({ id: User.id, token: "", error: "" });
    } catch (e) {
        return res.status(400).json({ id: "", token: "", error: e });
    }
});
```

Figure 5.2.6.J. Login Function

The create account function is straightforward, as it will take the appropriate user information passed in the request and store it in the database with a status return of 200. Any invalid parameters will be caught in the error and return 400. As we have not been given any other restrictions on account detail specifications, we have no specific errors to look for outside of empty fields, however we would implement such restrictions here for the most part.

```
router.post("/create-account", async (req, res) => {
    try {
        // validate data, create user, and save to database
        const User = await db.user.create(req.body);

        return res.status(200).json({ id: User.id, FirstName: User.FirstName, LastName: User.LastName, error: "" });
    } catch (e) {
        // catch any errors
        return res.status(400).json({ error: e });
    }
});
```

Figure 5.2.6.K. Account creation

The delete admin function simply needs an appropriate admin ID parameter and will destroy that element in the database, returning 200. Invalid IDs will be caught and return status 400.

```
router.put("/delete-admin", async (req, res) => {
    const {id} = req.body;
    try {
        // delete user based on id
        await db.user.destroy({where: {id: id}});
        return res.status(200).json({error: ""});
    } catch (e) {
        // catch any errors
        return res.status(400).json({ error: e });
    }
});
```

Figure 5.2.6.L. Delete User

Permissions can be updated for admin levels by reading in the admin ID and the requested level to be updated to. Currently, we only strictly need a proper administrator user within the scope of our customer's request, however this primarily fulfills their stretch goals of adding various user levels to allow different degrees of influence over content on the application.

```
router.put("/update-permissions", async (req, res) => {
    const {id, AdminLevel} = req.body;
    try {

        await db.user.update(
            { AdminLevel: AdminLevel },
            { where: {id: id}}
            )
        return res.status(200).json({error: ""});
    } catch (e) {
        // catch any errors
        return res.status(400).json({ error: e });
    }
});
```

Figure 5.2.6.M. Update user permissions

5.3 Database System

For our database system, we first looked at relational and nonrelational databases, and we eventually decided to use MySQL as our main system.

5.3.1 Relational vs Non-Relational

What is a Relational Database?

Relational Databases store data in tables and records (fancy word for rows).[34] Developed in 1970 at IBM, it works by linking information via the use of “keys”, which are unique identifiers for each record. These unique keys are referred to as “primary keys”, which can be used for multiple records in multiple tables.

There are also “foreign keys”, which is a primary key for another table stored in another. To give an example of that, if you look at Figure 5.3.A, we can see that the Employees table the primary key in this case is EmployeeId. However, the relation between Sales and Employees is EmployeeId as the Sales table has a foreign key of EmployeeId. This means a sale cannot be made without a valid unique employee. Just as in the real world, there are no sales without the help of an employee to sell it to the customer.

In addition, when a record with a relational primary key is deleted/updated, we need to delete/update all the references to that primary key as well, which can be accomplished by using a cascade delete/update.

Finally, querying the data is done by using Structured Query Language, otherwise known as SQL. SQL can perform CRUD operations on tables but heavily relies on the use of keys for storage and search.

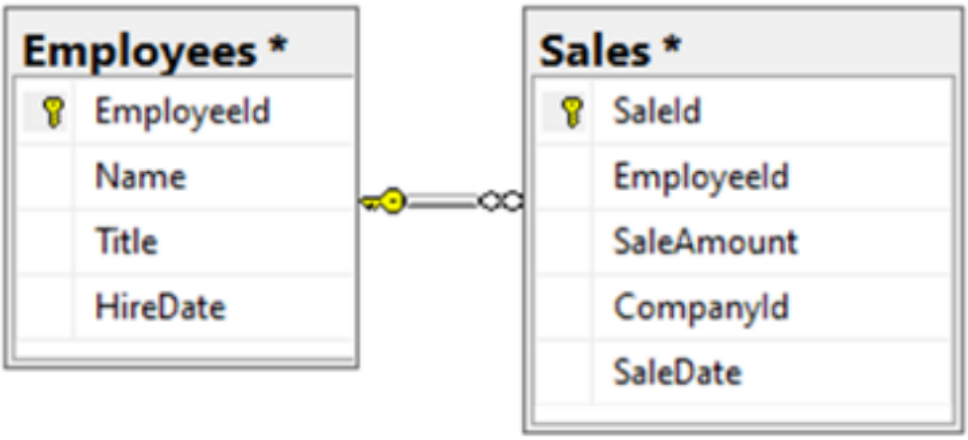


Figure 5.3.A. Entity Relationship Diagram

What is a Non-Relational Database?

Non-Relational databases known as NoSQL, have no rows or keys. Alternatively, it used optimized storage models based on the data being stored.[35] Storage models can range from Documents, Columns, Key-Value, and Graph databases.

Overtime, these get very complicated not only to maintain but understand parts of your database vs a traditional structured relational database.

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Figure 5.3.B. Traditional Relational Database

5.3.2 What is MySQL?

When considering the options for our database system, it was crucial to our customer that one of the requirements must be that the backend was going to be MySQL. MySQL is a relational database management system (RDBMS) developed in 1994, which was later bought out by Oracle in 2010[36]. MySQL databases are relational and contain a set of structured/defined tables that are like Microsoft Excel. These tables help optimize actions and provide high performance for an industry standard that allows rapid development for developers while giving users a friendly experience when performing querying.[37]

5.3.3 Benefits of MySQL

- Data Security
    - MySQL is one of the most secure database management systems and used in popular web applications such as Facebook, Twitter, and Instagram. The data security encryption protocols such as Secure Sockets Layer, data masking, and firewalls help prevent cyberattacks.

- High Performance and Scalability
    - MySQL was developed for speed and when increasing the number of Users or Letters in this project, the speed won't drop off and cause any overall performance issues.

- Relational Database
    - When keeping track of metadata such as coordinates, dates, locations things could get very messy in a Non-Relational Database. Thus, it is crucial to keep all of this data uniform for not only performance reasons but to maintain good programming practices.

- Open-Source
    - MySQL is free and gives the users the ability to tailor its source code to fit requirements. Given this flexibility, users may find ways to improve and secure MySQL more, which can then be easily upgraded without disrupting any services.

5.3.4 MySQL Components

Below are the multiple components of MySQL. These features will be explained in detail along with their purpose. [38]

- Thread Handling
    - Manages the established connections on a thread

- Parser
    - Breaks down the data into several tokens and uses it to create a data structure from entered data

- Optimizer
    - Provides query rewriting and improved table scanning for better performance

- Buffer and Cache
    - Stores commands and results of queries. Prior to querying it searches the cache to see if the result already exists.

- Table Metadata Cache
    - Columns and information about the tables

- Key Cache
    - Entry made in the index that helps identify certain parts of the cache

5.3.5 MySQL vs Other Database Systems

When gathering requirements, our customer made it clear that one of the key requirements was the backend system design to include MySQL. In addition, the advantages of using MySQL versus other database systems include performance, structure, and open-source abilities.

An option that was considered was another relational database such as PostgreSQL, however with the tables being created for this product, object-oriented design was never really a necessity despite PostgreSQL having the ability to store more data types such as tuples, maps, etc.

MySQL vs PostgreSQL

Before we discuss the pros and cons of PostgreSQL, we will discuss PostgreSQL's history along with some of the components it offers.

What is PostgreSQL?

PostgreSQL (otherwise known as Postgres) is another open-source relational database management system[38]. Created in the 1980s, it is known to be a database used for a LAPP stack (Linux, Apache, PostgreSQL, PHP).

PostgreSQL is great for unstructured data and object oriented design. It features a wide variety helpful feature such as:

- Ability to define your own data types
    - This helps give that object oriented design for tables. We can then begin to perform some polymorphism and inheritance amongst sub tables.
- Views
    - This is a way for users to present their data in a different way for users. For example, let's say we need a set of values from both table A and table B. In MySQL, we query for this data in table A and table B, but in Postgres with the help of views, we can join tables to create a view (table AB) and query them with ease.
- Subqueries
    - We can perform nested query options as such below.

```
SELECT inventory.film_id FROM rental
                INNER JOIN inventory ON inventory.inventory_id
rental.inventory_id WHERE return_date BETWEEN '2005-05-29' AND
'2005-05-30';
```

MySQL vs PostgreSQL

PostgreSQL offers a wide variety of features. It tends to have performance issues when comparing it to the Figure5.3.C[39]. PostgreSQL is great for handling large datasets and performing read-write operations such as altering tables, creating views, etc. However, MySQL remains superior in read-only procedures such as SELECT. With this, we concluded although PostgreSQL has great functionality with its object oriented design, it is more important to read our map data in an optimized manner.

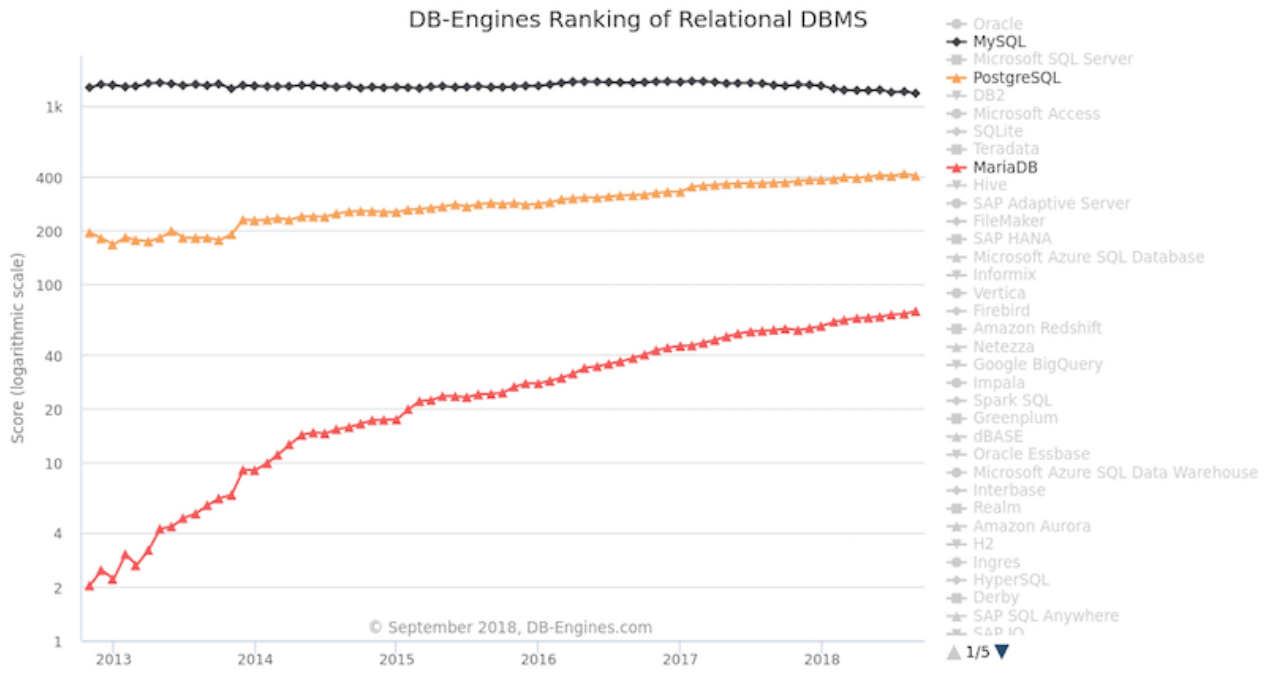


Figure 5.3.C. Ranking of Relational DBMS

MySQL vs MongoDB

Furthermore, MongoDB's non-relational database system does not fit this requirement of structured data. A lot of this metadata is consistent and will be labeled accordingly during database ingestion, so it seems only to make sense to use MySQL, which takes advantage of structured datasets[40]. In addition, considering the scalability of MySQL in structured data, it is evident that it has far more advantages than MongoDB. Therefore, we did not see MongoDB as a viable solution for our backend database system.

5.3.6 MySQL Commands

The following is a summary of some important MySQL commands that will be used during the development of Mapping Keats.

1. `SHOW TABLES;`
    - Shows us the list of tables we currently have in our database

2. `CREATE TABLE [IF NOT EXISTS] table_name(column_list);`
    - Creates a table with a list of columns that does not already exist. We will use this primarily for creating the Users and Letters table.

3. `SELECT * FROM table_name;`
    - Show all rows and columns from a table.

4. `SELECT select_list FROM table_name WHERE condition;`
    - Show all rows and columns from a table that satisfies some condition (e.g ID = 5).

5. `INSERT INTO table_name(column_list) VALUES(value_list);`
    - Insert a new row entry into a table.

6. `UPDATE table_name SET column_1 = value_1, ... WHERE condition`
    - Update row(s) in a certain table that satisfy some condition.

7. `DELETE FROM table_name WHERE condition;`

   · Delete row(s) from a table that satisfy some condition.

8. `SELECT select_list FROM table_name WHERE column LIKE '%pattern%';`

   · Search for data using the LIKE operator to give us back relevant results

6 System Design

This section contains an in-depth discussion of the system design of our product, which is divided into numerous sections.

6.1 Overview

When creating Mapping Keats, we chose to split the development of the website into multiple sections that each hold a key piece of functionality. This allowed each member to focus on one section at a time, while other members worked on their assigned areas. Each section is led by one of our team members, shown below:

- Admin System
- API System
- Querying System
- Map Plotting Visualization System

In addition, below are high-level summaries for each section we will discuss.

6.1.1 Admin System

The admin system will be used for specified admin accounts to perform operations on John Keats letters behind the scenes. It will be composed of a data management system that allows admin to add, edit, and delete John Keats letters. In addition, admin will have the ability to upload letters through a .csv/.xlsx upload ingestion process so large amounts of letters can be easily added with ease.

6.1.2 API System

The API system will be the component directly responsible for making the following features available to our admin and users:

a. Database queries and edits

b. Map plotting and data visualization

c. Admin logging in

d. File upload ingestion process

e. The flow of information between the frontend and backend throughout this project

6.1.3 Querying System

The querying system is one of the core components used for admin and users to do search operations. The querying system can be used to search for things such as:

a. Keywords

b. Recipients

c. Location

6.1.4 Map Plotting Visualization System

The map plotting system is also one of the major components and requirements for this product. The map visualization will have:

a. A heat map of all the letters in their locations

b. A temporal time slider that animates where the letters travel over a period

c. Visualization of letters on a historical map

6.1.5 Wireframes

To better communicate with our sponsors and ensure the project is meeting their vision, wireframes became a necessity. They are quick to learn how to do and faster than programming to implement, allowing us to get feedback and make changes quickly.

Initial Wireframes

All of our wireframes were made in figma[42]. We came up with this initial wireframe pretty early in the development. Just to convey some information about the project and communicate better between our teammates. This wireframe has basic functionality and tools for this project. And then after a couple of meetings with both group members and with the sponsors, we expanded on the wireframes and added more functionality and tools.

Why Figma?

Before any actual webpages are created and hosted on the server, an initial design must be agreed upon by the project team and its sponsors. To express the early ideas of what these pages might look like when they are finally written in code, our team has utilized Figma, which is an interface design tool. Figma allows our team to create the multiple pages that our website will have without doing any coding. It is like graphically designing our website and its components, except we can also include small amounts of functionality to these designs, giving the team and our sponsors a feel for how the website will work before any programming has commenced. Once we design enough pages, we can utilize some of the functionality Figma gives us by allowing buttons on the page to “redirect” the viewer to other pages accordingly. This is very helpful when presenting design ideas that may not be all that similar to each other, because one design may use buttons for all functionality, when another may utilize swiping, pinching, or other maneuvers. The viewer gets a much better idea of how each design works and allows them to make a better decision on how they want their product to look and feel. [43]

Figma is a great tool because we can experiment with many different layouts, color schemes, fonts, and any other front end components that will be required. These interfaces can be designed much easier and faster than experimenting with the actual code, and allow the sponsors to agree on a certain design schema so that when it comes time to implement the design on the front end, we know exactly what needs to be done and how it should look at the end. Figma is also collaborative, allowing all the team members to view and contribute to the design where they see fit. This allows the entire frontend team to work on different parts of the same project, as well as take any immediate feedback from our sponsor. Figma's website also allows for a number of plugins, allowing the designer to create anything they may need, just like how programming enables the implementation of any service the client asks for.

One of the most useful features Figma offers is the ability to translate your design directly into code. This means when the final design has been agreed upon, the team can mimic all the components to an exact copy in their code. Clients do not have to worry that the final product might have small differences here and there, and developers do not have to stress over trying to get small things perfect. Figma enables our team to create the exact product that was shown to the sponsors which will make them happy with the work we have done.

6.1.6 Overall Design

In this section, we will cover our overall design of this product by using a use case diagram along with a sequence diagram to further explain our system design at a higher level.

Our use case diagram in Figure 6.1.6.A shows that the user and admin contain similar operations. However, only an admin has the ability to create an account and perform CRUD operations on the letters. The user can only query and plot letters for data visualization.

The sequence diagram in Figure 6.1.6.B displays some of the most common actions that users will take on the website. Its purpose is to show how an action from the user

causes the website to interact with the front-end, API, and database. Certain checks are made by the API before any changes are made to the database, and the API can communicate any errors or results to the front-end.

Mapping Keats Use Case Diagram

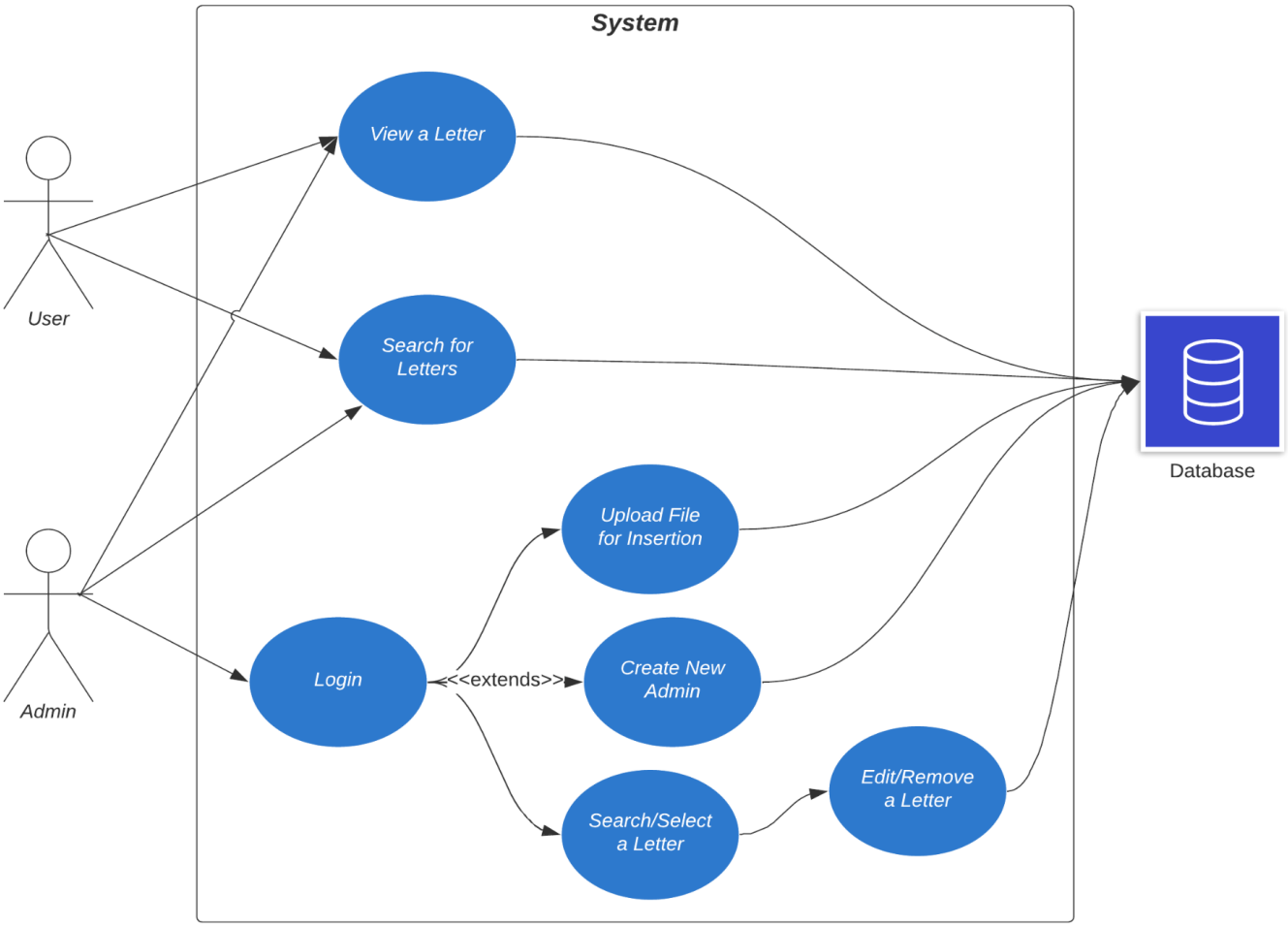


Figure 6.1.6.A. Use case diagram

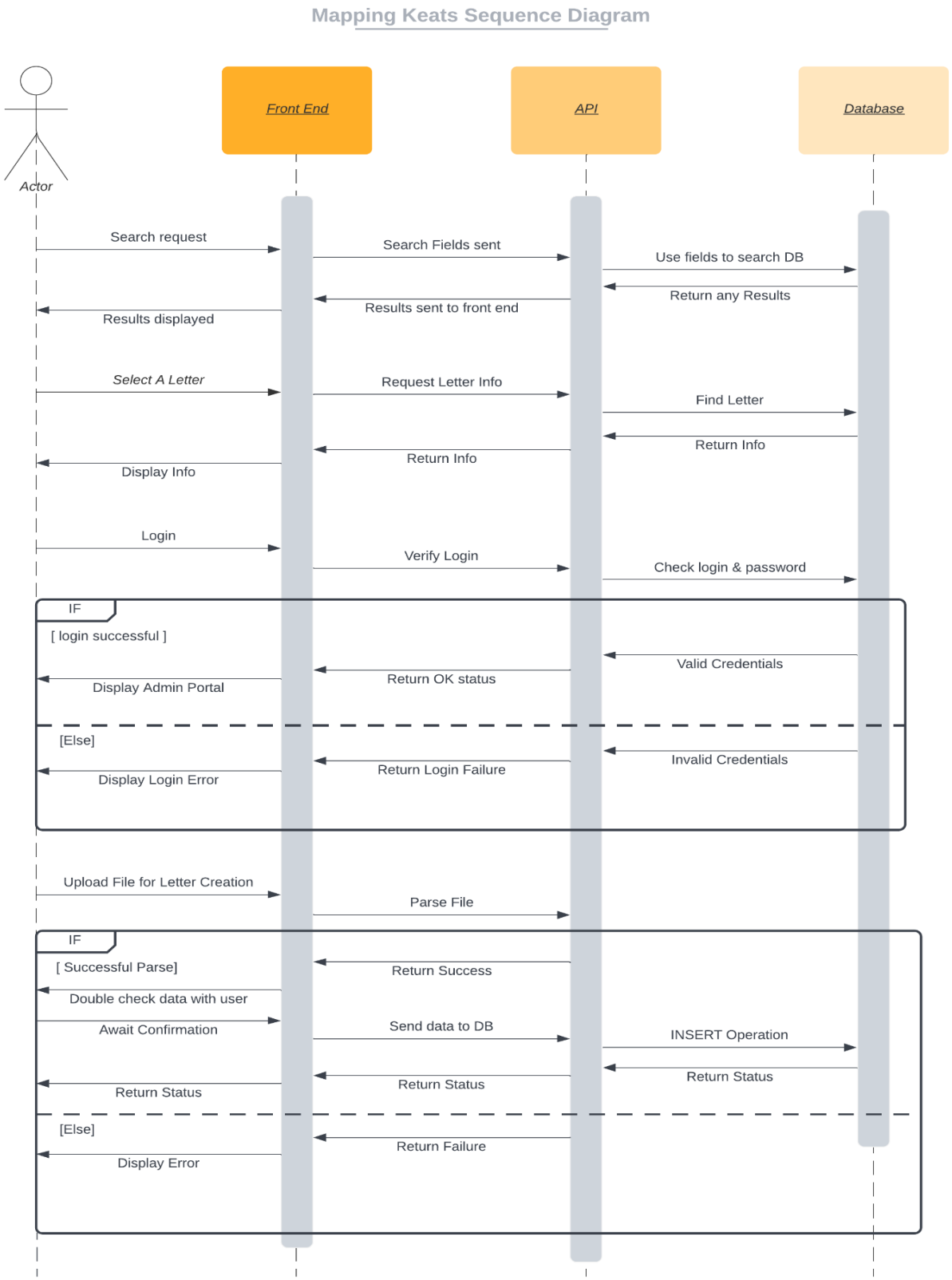


Figure 6.1.6.B Sequence Diagram

6.2 Front-End

The frontend is composed of the admin system, querying system, and the map plotting visualization system. In this section we will go further into details about their respective designs along with showing some initial/final wireframes.

6.2.1 Admin System

Login page

This page, shown in Figure 6.2.1.A, is fairly simple but ultimately very important. It allows a user or admin to login and gain access to their allowed viewings. At first only the super-admin will have a prebuilt account and be able to perform this task, but they can add other accounts at specified levels (admin, user, etc.). The navigation bar at the top right allows for quick access to our home and about page. There is also the option of password recovery by selecting “Forgot Password?”. Each account will have an associated email where the recovery password can be sent to and reset to a new user specific password. After login, a captcha test will be applied for when this web app is made public, shown in Figure 6.2.1.B.

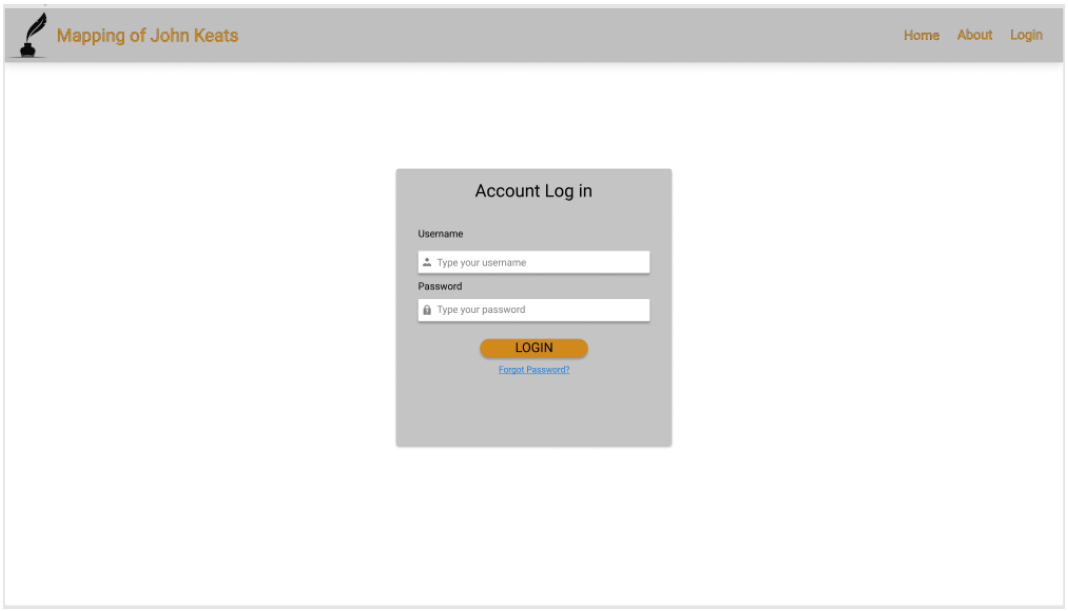


Figure 6.2.1.A. Account login page with forgot password

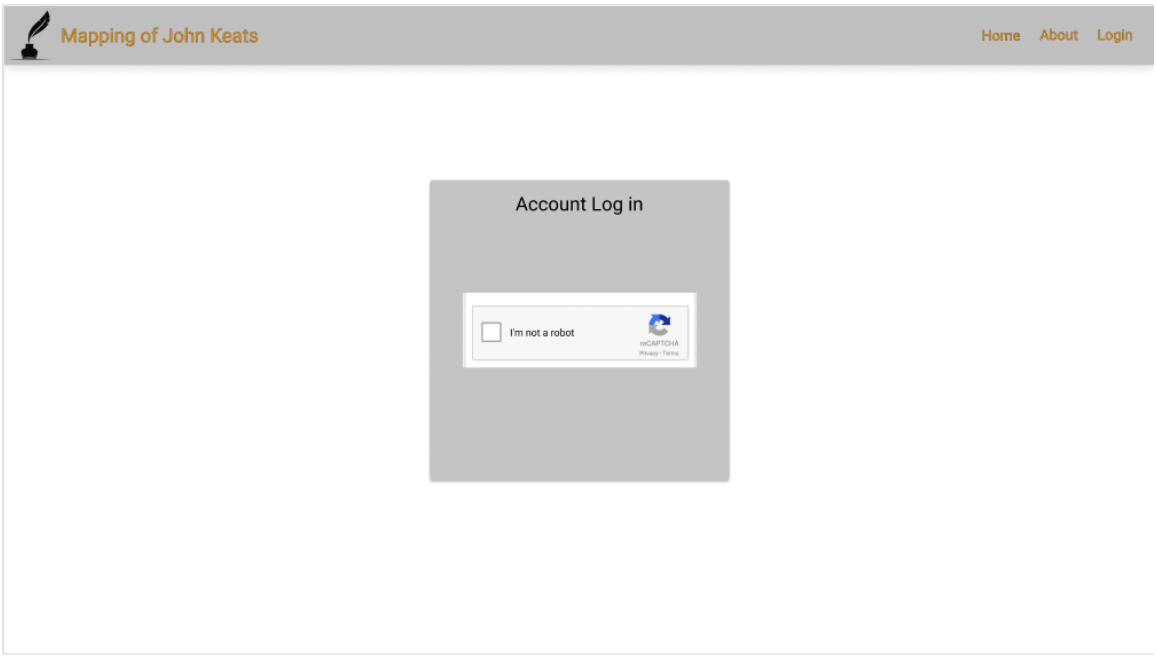


Figure 6.2.1.B. Login Captcha test

Admin page

This page, shown in Figure 6.2.1.C and Figure 6.2.1.D, is only accessible by the admins and super admin. It includes the standard navigation bar at the top and several new options on a sidebar.

1. Account. Under the account tab, the user can view their account settings such as username, email, and password. They can change these settings.

2. Add User. The admin can add another user account while the super admin can add user and admin accounts.

3. Database dropdown. This is where the user can perform database CRUD operations. The first page would be the search ability. They can search the database for specific entries and alter or delete them. The add tab allows them to insert a single entry into the database. The user then either saves the changes made, or discards them, finalizing the database entry. The import tab will allow the admin to import several entries into the database using a standardized.

a. Search Figure 6.2.1.C. Within the search tab of the admin page, the user can locate existing entries using a variety of methods with the results showing up at the bottom. The results can then be selected for different purposes. They can either be viewed, edited, or removed. Viewing the letter will bring up a page that contains the letter in its entirety. Edit will allow the admin to change information currently stored at that letter. Remove will completely remove this letter entry from the database.

b. Add Figure 6.2.1.D**.** This is where admins will add new letters to the database. They simply need to drag and drop .xslx files in and they will be added. The required format of the. xslx will be provided as a template for the user to look at and copy. At any point while selecting the desired files, the user can either deselect one or more files to be uploaded or clear the entire current selection. Once the user has selected all the files they wish, the upload button will add the entries to the database.

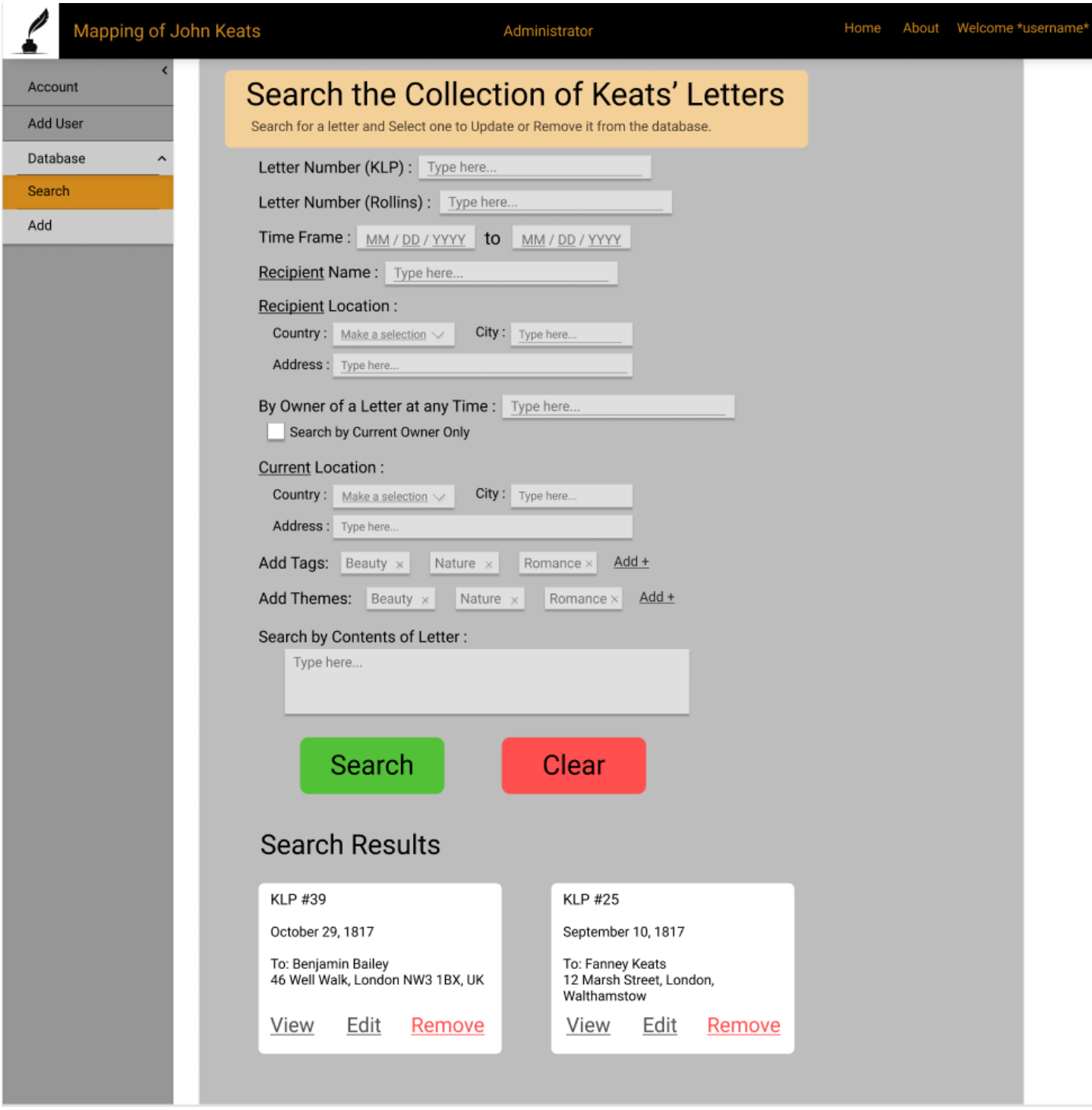


Figure 6.2.1.C. Admin Search and Edit Letters

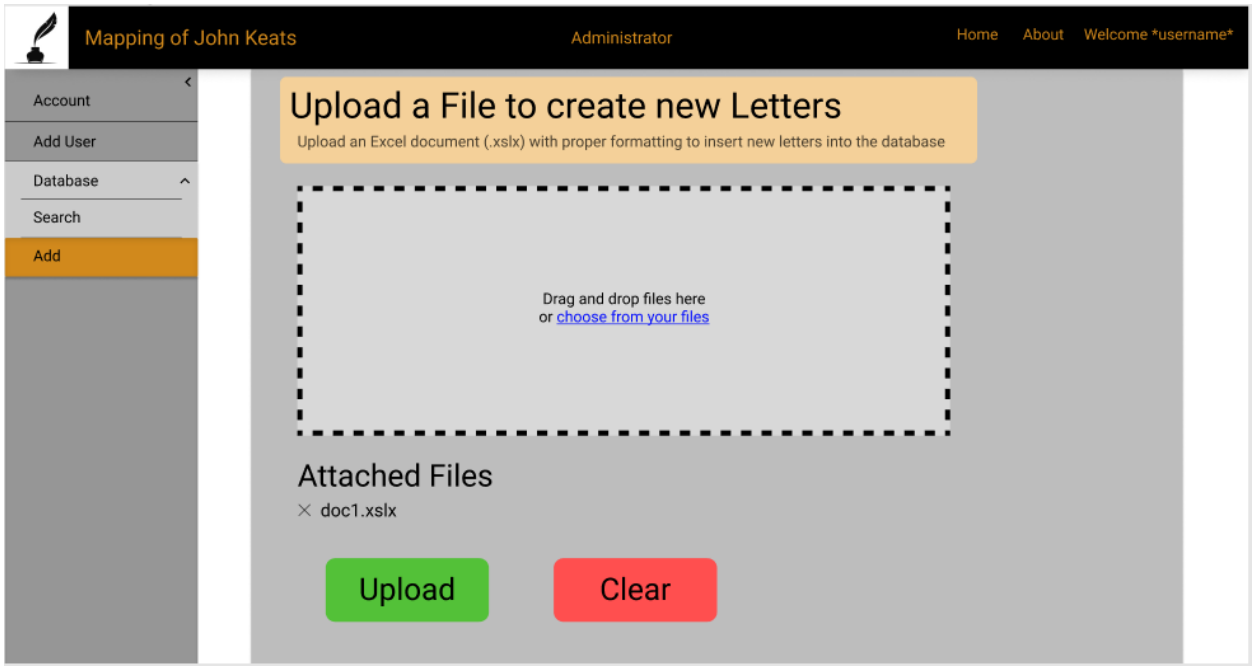


Figure 6.2.1.D. Admin add new Letter(s) Page

6.2.2 Querying System

Part of the functionality of this website will be a query system so that users can view a subset of letters based on a number of different fields. If a user wants to find one letter in particular, view a collection of letters in a certain time frame, locate letters that mention certain words or are written with a unique theme, this query system will give the user the power to do so.

Popular Search Fields

There are many fields the user can search by, but some of the most popular fields for this query system will be the recipient of the letter, the date when it was sent, the city or country where the letter was sent, and words or phrases that are contained in the letter. Information about the recipient(s) of the letters and their location are stored as well,

meaning users can pull up all letters sent to a specific person, and even further can search for that one person at a specific location if desired.

Another desired field will be searching for letters sent on a particular day, or a range of dates. One feature that will be helpful for the visualization of this data is that when users search for a range of dates, all letters that are returned will also be mapped to a temporal slider, allowing the user to slide through time and see exactly when and where those letters are sent.

Locations will also be stored in two formats: street address and geographic coordinates. On top of these two location formats, we think it would also be quite helpful to include certain location related keywords into the tags section for that letter enabling the letter to be returned in broader searches. For example, letters sent to addresses inside England are written as “UK” on the address, so including a tag for “England” or “Great Britain” would increase accessibility for users that are not knowledgeable about the correct terminology, as well as any language differences between regions.

This application will also hold an image of the original letter as well as a transcript of the message, which will allow for searches on the actual text of the letters. Many letters contain repeated phrases or vocabulary unique to John Keats, and this enables users to study the vocabulary he used in his messages.

Provided is an extensive list of all popular search fields:

- Date of when letter was sent(particular date or a range)
- Location
    - Street Address
    - City or Country
- Letter Recipient
- Text Search

Secondary Search Fields

Some of the other fields that can be used for searching are people or organizations who currently own the letter, location of the current owner, the unique number for that particular letter (out of 252), event descriptions for each letter with an according date, tags for each letter which can contain various keywords, and themes that relate to the content of the message.

Since these letters were sent 200 years ago, letters are no longer owned by the original recipient. Users may want to search for letters owned by a certain person or location, and even where they might currently be held. Each letter has a name and location of the current owner allowing for these searches to be made.

All letters have two unique ID numbers associated with it as well. One is the Keats Letter Project number, and the other is the number given to it by Hyder Edward Rollins, a renowned authority on John Keats and his collection of letters. If users want to find one or more letters without a broad search, they can do a quick lookup with these numbers.

Descriptions of events related to each letter are also stored in the database. These events are things like when the original letter was sent, postmarked, received, where the recipient took the letter, when and where the recipient moved, who it was secondarily sent to, and other events relevant to the letters history.

Tags will also be assigned to each letter, allowing users to search for keywords that relate to the letters, but might not be in the address or transcript. These tags include words like synonymous location names, topics that were written about, names and locations that have some relevance to that letter, and much more. Similar to how we keep a list of tags for each letter, we store a list of themes for a particular letter as well. When Keats wrote each letter, each one carries certain themes and many literature students enjoy studying these recurring themes. For example, one may wonder if his letters written near the end of his life hold any common themes that did not show up in previous letters. This field will enable users to make searches as such.

Provided is a list of all secondary search fields:

- Current Owner
    - By Name
    - By Location

- Keats Letter Project Number
- Rollins Letter Number
- Letter Description
    - Typically Key Events
- Tags
- Themes

6.2.3 Map Plotting Visualization System

Map page

The map page is where a user will spend most of their time (Figure 6.2.3.A and Figure 6.2.3.B). All of John Keats letters will be displayed on the Leaflet map on this page. A summary of features going from top left to right before going down a layer and restarting at the left:

1. Navigation bar. As with the previous pages, there is a navigation bar at the top right with quick access to home, about, and login. There is also a “map options” toggleable button there. This brings up and closes the map searching popup on the map.
2. Map. The map includes buttons to zoom in and out as well to make the map full screen or not. The actual map will be draggable with the mouse cursor and cropped with the mouse wheel. On the map will be situated POI markers that indicated one of John Keats letters was or is there. The map will contain a scale at the bottom left that scales appropriately to the current zoom level.
3. Legend. The map legend will be collapsible and draggable so that it does not get in the way. On it will be important information about icons on the map that may not be immediately understood. For example, it states what the different colored lines mean (red for where the letter has been and blue for where it went after).
4. POI markers. Each marker can be clicked on. When done so, a popup will appear right next to it. Clicking again will close this popup so multiple POI’s can

be viewed and compared at once. This popup will display a lot of information about the letter located here. This information is: Location, where it just came from (display origin if this is its place of writing), origin (if not already there), where it's going next, current date, who wrote it, and important tags. There are also two important buttons that will open up a side panel of the page that would display additional information such as the original letter image or a transcript which is shown in Figure 6.2.3.B. The last thing in the popup is a toggleable button labeled "Toggle Path". This will cause lines on the map to show the path this letter has and will travel. Red lines will connect it to locations it's already been while blue lines will connect it to locations it will go to next.

5. Map options. This is where the user will be doing all the information filtering. This window can either be automatically kept open or closed when applying a new filter, user choice. The first filtering option is selecting time. The user can choose to either only show results from a specific day, or a range of time. If selecting a range of time, a temporal slider can be enabled allowing the user to quickly glide through the time range. They can also snap to the currently viewed POI marker. This is useful if they've moved their screen to a different part of the map and want to quickly go back to the marker. They can search for a specific location and have the map view shift there. Each letter has a series of tags, such as death and marriage, associated with it and the user can filter the letters by these. While modern technology has made mapping incredibly accurate, this project requires that information can be viewed on time accurate maps. In the layer type drop down the user can select which map layers are applied. Options include geo-rectified maps applicable to the current time period, a heat-map, or a dark-mode option. The last option is a randomize button. Through sponsor request, this button will select random filter variables that at least displays one POI.

6. Side panel. This side panel, shown in Figure 6.2.3.B, is opened within the POI markers. This panel can have several tabs within it depending on what information this POI marker contains. The project is being developed using "The Keats Letters Project"[3] which is a collection of all John Keats letters. Each entry has at least a paragraph talking about it, the Overview tab is where this information goes. If a particular letter has a typed transcript of it, it will go into the Transcript tab. Most letters do not have a transcript, only containing an image of its handwritten form. These images will be within the Image tab. Brian Rejack has a "This Week in Keats" video series that applies to a few letters, the Video tab will house these.

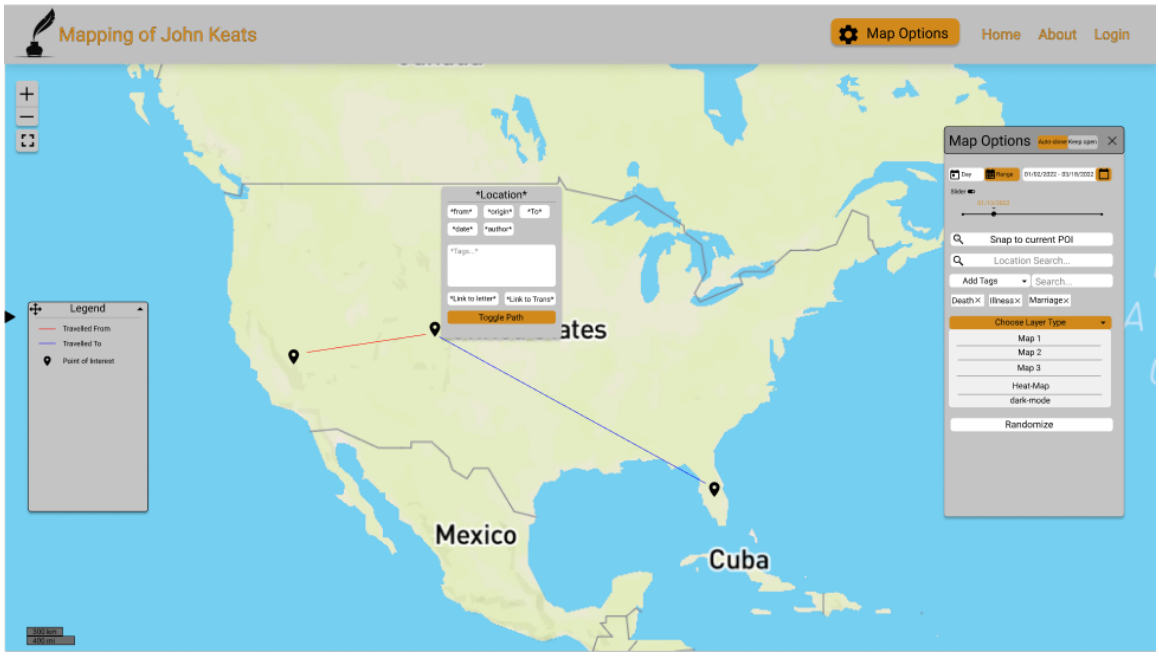


Figure 6.2.3.A. Map page with PoI selected

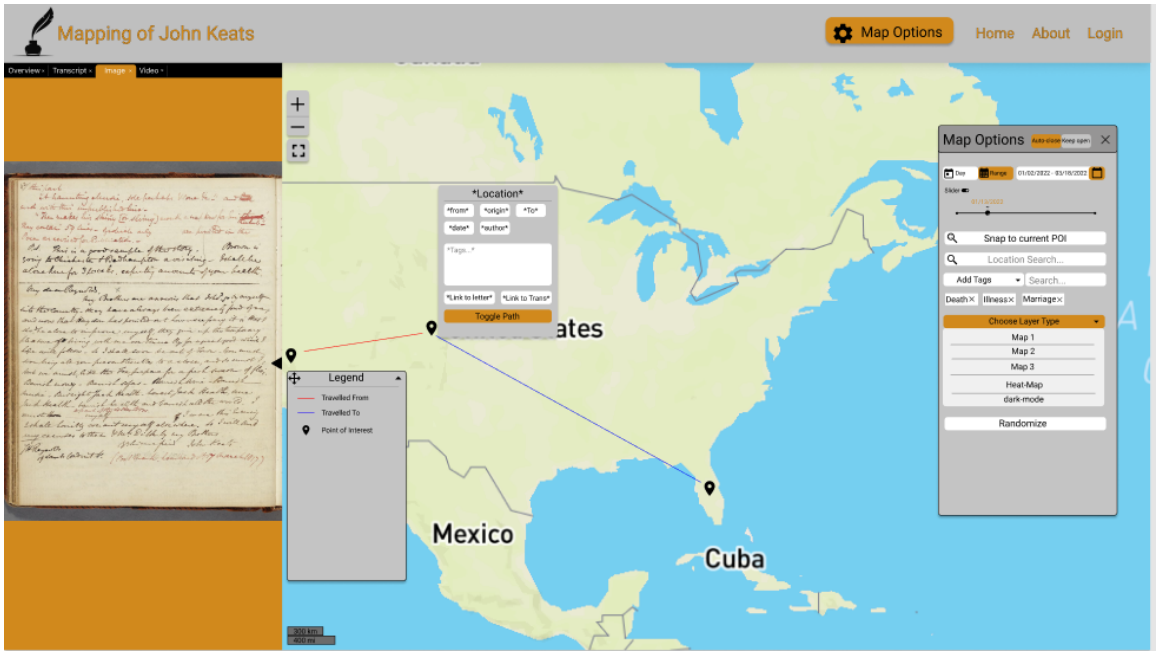


Figure 6.2.3.B. Map page with sidebar info open

Home Page

The home page includes basic information about John Keats and his letters as well as a brief section on how to use the map on the Map page. It is accessible to anyone whether they are logged in or not. It also includes a navigation bar at the top right for quick access to the about page and the login page. It is from this screen where a user gains access to the map page.

About Page

This page could either be an about page for John Keats, where we would give information about him and his life or could be an about page for introducing the project, members, and give a general idea behind this project. Former would give more recognition to John Keats and could give more context to the letters and that could be very important to the readers. The latter could give the readers an overview about this project and that could be also valuable information. Or we might combine those two ideas for the about page as well. Either way an about page we think is appropriate for this project.

Letter Archives Page

This page could be used for keeping the letters of John Keats in a page of itself where users can scroll down and see the pictures of the letters, and as well as the text version of the letters. This page might be good for the users that don't know what to search and give them another option to just scroll down all the letters then choose whatever letter they like.

Admin Login

After the Letter Archives button, there is an Admin Login button. When clicked a drop down of username and password forms pop up as we can see on the second page. It's a very simple looking form for now but since it's a prototype we will be adding more styling and more features like password reset, and forgot password options.

Figure 6.2.3.C. Homepage

Data Visualization

After the user has made a query for certain letters, we must present those letters in a way that is easy to understand and interact with. The two main ways we will do this is with a heat map, and individual letter mapping. The heat map, as shown in Figure 6.2.3.D, will be useful when the query has returned many results, showing hotspots of collections of letters on popular locations with a respective size and color. Users will be able to click on each hotspot to see details and have the option to select one letter for further analysis. Individual letter mapping, similar to Figure 6.2.3.E, will be used when only a few letters are returned from the query, or when they are looking at just one letter at a time. Each letter will have an origin and a destination, and the map will be resized so that the letter's path does not go off the screen, or only span a few pixels. We will be using Leaflet, a javascript library to assist in the visualization of this data on different period maps.

When trying to visualize a large collection of Keats' letters, we face the challenge of finding the best way to display those letters so that we provide the user with enough data, but not so much that it looks messy and confuses them. A heatmap does a good job of showing where the letters were concentrated, but lacks when the user wants to see where those letters were sent. Individual mapping of each letter does the best job, showing where each letter originated and was sent to, but suffers when the user is trying to view many letters at one time.

The solution that was presented to our team by our sponsor was to create a temporal slider so that letters appear and disappear when dates are passed over by the slider. This does a great job at visualizing a large set of letters without drawing so many lines on the map that the user can no longer follow along. When we are displaying all the letters at once, or even a range of dates given by the user, the website will present a slider that allows the user to scroll through time and see when letters are sent and received, as well as map other events that are given in the description for each mapped letter. Users will have the option to select which method they view the letters in order to give them the best experience on our website. [43] [44]

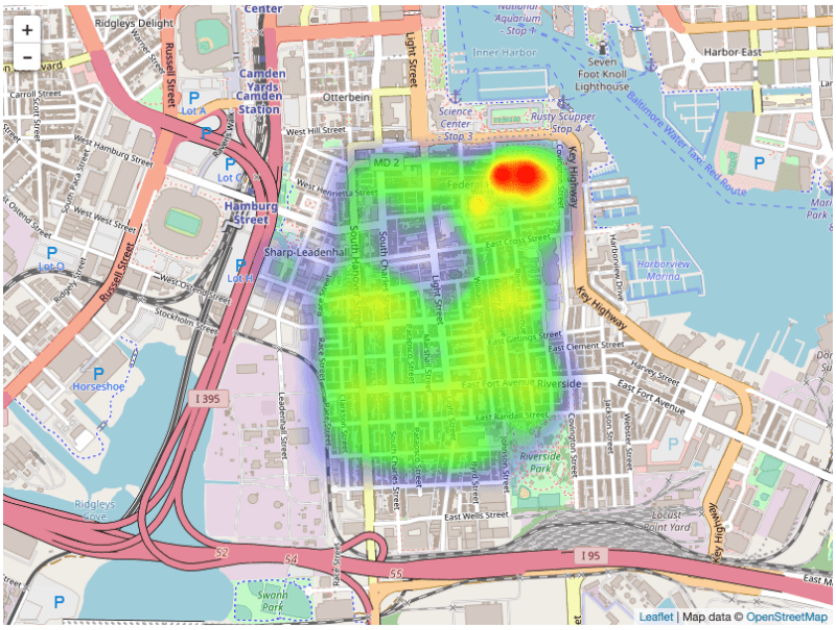


Figure 6.2.3.D Heat Map Example

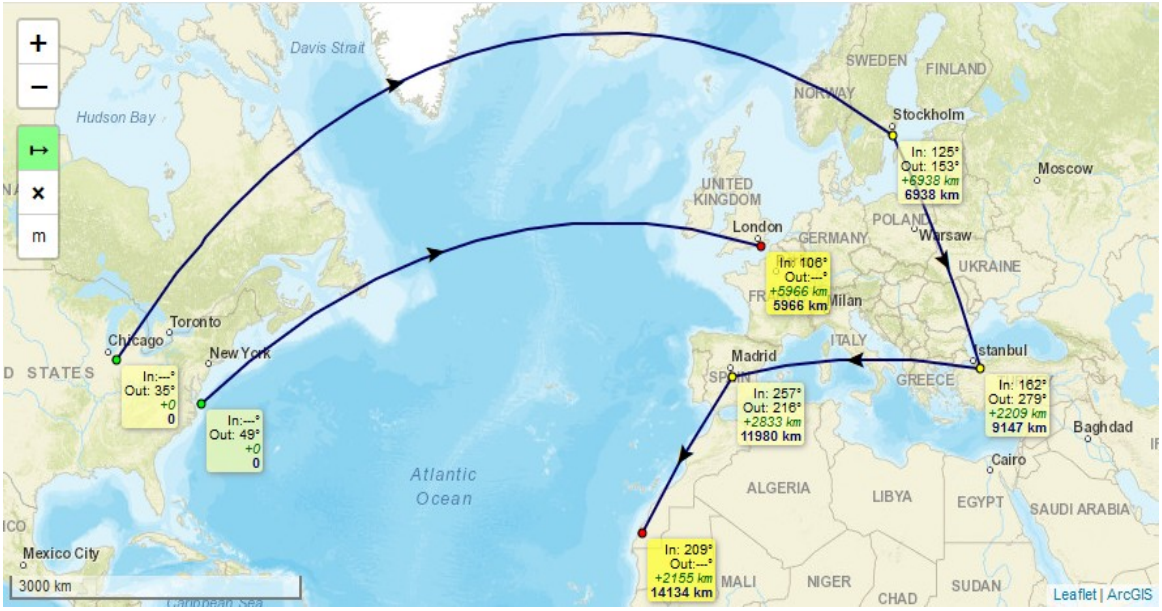


Figure 6.2.3.E PolyLine Map Example

6.3 Back-End

After researching various back-end solutions, we decided to go with Node, Express and Sequelize. In this section, we will further explain our implementation thus far of File ingestion along with some Admin routes.

6.3.1 Data Upload/File Handling

As time progressed with this project, we needed an efficient way to handle spreadsheets full of metadata to be uploaded into our database. Whether it be John Keat or another author, there had to be some friendly fashion to ingest any spreadsheet and handle on the backend, so the admin does not have to go through the hassle. Here are two solutions we came up with during this process:

1. Creating an API route called /upload-letters, where the user can throw their spreadsheet data into a python script, and it is posted to the database. However, our API will not be public in this case and this requires our customer to get their hands dirty during the upload process.

2. A friendly UI in the admin portal where the user uploads a .csv or .xlsx and the API/Backend takes care of the ingestion process. The only downside to this is if the admin has bad formatting on the spreadsheet, it may have some bad effects on the overall app. This could potentially be handled with a sanity check prior to upload, however this may not be a viable option at a mass scale since it could be hard to find where in the spreadsheet caused the error.

6.3.2 User Routes Implementation

By utilizing Express and Sequelize, we were able to properly implement create, update, and delete functions, as well as basic login, for admin users in the application. The following libraries were implemented in our routes in Figure 6.3.2.A.

```
const router = require("express").Router();
const db = require("../models");
const bcrypt = require("bcryptjs");
const dotenv = require("dotenv");
const { Op } = require("sequelize");
```

Figure 6.3.2.A. Libraries

The login function is as follows in Figure 6.3.2.B. Some elements of note include the usage of bcrypt to hash the password. Users are searched by the email and password parameters passed in the request. Invalid users will successfully receive a 401 error, otherwise a successful login will return status 200.

```
router.post("/login", async (req, res) => {
    const {email, password} = req.body;
    let hashedPassword = bcrypt.hashSync(password, 10);
    try {
        // find user
        const User = await db.user.findOne({
            where: { email: email, password: hashedPassword },
        });

        if (!User)
            return res.status(401).json({ id: "", token: "", error: "Invalid Credentials" });

        return res.status(200).json({ id: User.id, token: "", error: "" });
    } catch (e) {
        return res.status(400).json({ id: "", token: "", error: e });
    }
});
```

Figure 6.3.2.B Login Function

The create account function is straightforward, as it will take the appropriate user information passed in the request and store it in the database with a status return of 200, as is shown in Figure 6.3.2.C. Any invalid parameters will be caught in the error and return 400. As we have not been given any other restrictions on account detail specifications, we have no specific errors to look for outside of empty fields, however we would implement such restrictions here for the most part.

```
router.post("/create-account", async (req, res) => {
    try {
        // validate data, create user, and save to database
        const User = await db.user.create(req.body);

        return res.status(200).json({ id: User.id, FirstName: User.FirstName, LastName: User.LastName, error: "" });
    } catch (e) {
        // catch any errors
        return res.status(400).json({ error: e });
    }
});
```

Figure 6.3.2.C. Account creation

The delete admin function in Figure 6.3.2.D simply needs an appropriate admin ID parameter and will destroy that element in the database, returning 200. Invalid IDs will be caught and return status 400.

```
router.put("/delete-admin", async (req, res) => {
    const {id} = req.body;
    try {
        // delete user based on id
        await db.user.destroy({where: {id: id}});
        return res.status(200).json({error: ""});
    } catch (e) {
        // catch any errors
        return res.status(400).json({ error: e });
    }
});
```

Figure 6.3.2.D. Delete User

Permissions can be updated for admin levels by reading in the admin ID and the requested level to be updated to as in Figure 6.3.2.E. Currently, we only strictly need a proper administrator user within the scope of our customer's request, however this primarily fulfills their stretch goals of adding various user levels to allow different degrees of influence over content on the application.

```
router.put("/update-permissions", async (req, res) => {
    const {id, AdminLevel} = req.body;
    try {

        await db.user.update(
            { AdminLevel: AdminLevel },
            { where: {id: id}}
            )
        return res.status(200).json({error: ""});
    } catch (e) {
        // catch any errors
        return res.status(400).json({ error: e });
    }
});
```

Figure 6.3.2.E. Update user permissions

6.3.3 API Endpoints

In this section, we will cover the API endpoints for the admin, querying, and map plotting visualization systems. Each endpoint will discuss the implementation of the route along with the process of handling this information.

Admin API Endpoints

The admin system is primarily containing a data ingestion upload process as well as your standard administrative operations. These are outlined below:

1. login (POST request)

    · This endpoint will be a POST request. This means that this will be used when an admin is trying to login.

    · Upon entering their login credentials and hitting the login button, we query our Users table to see if there are any matching login credentials. If there is a match, we send back the UserID and information about that admin to be handled on the client side. Otherwise, we send an error message to be displayed.

    · After this depending on which response code, we send back, we either advance from the login page to the admin portal or we display a message stating “Invalid Login Credentials”

2. create-account (POST request)

    · This endpoint will be a POST request. It is used to create a new admin account.

    · The super admin only has access to this functionality in our web application.

- The super will enter the new admins name and email address in addition to what admin level permissions they are granted.
- From there, we check the database to see if that email already exists. If so, we resend an invitation email. Otherwise, we will create that user with a temporary password and send them an email to join.
- Finally, we will send our response back to the front-end saying whether the operation was successful or invalid.

3. update-permissions (PUT request)
    - This endpoint will be a PUT request. It is used to update admin permissions and can only be accessed by a super admin.
    - On the front-end, the super admin will see a list of the admin accounts displaying their permission level in a dropdown. From there, the super admin can edit permissions for each admin.
    - If there is a change in the dropdown option, an onChange event occurs and will update the permissions related to that admin account

4. delete-admin (PUT request)
    - This endpoint will be a PUT request. It is used to delete an admin account and can only be accessed by a super admin
    - A super admin has the option to delete an admin from the portal. When selecting the admin on the front end, we ask them a confirmation message to make sure they are sure, and then we proceed to delete them
    - On the server side, we delete that admin user by their ID and return a success message for the client side to handle.

5. Upload-letters (POST request)
    - This endpoint will be a POST request. It is used to upload letters to the database using a .csv/xlsx file ingestion process. This operation is only available to admin users.

- The admin uploads a properly labeled .csv/.xlsx on the upload page and from there.

- Once the file has been received, we open the file and perform a sanity check to make sure there are no errors such as mislabeled columns, duplicates, or conflicting entries. If the file has any errors, we report the error back to the front end, where the user will have to go through and update the file. Otherwise, we can perform an ingestion.

- During ingestion, we go row by row and create new Letter entry objects and save them to the database using sequelize. After this has been successfully completed, we report back to the front end with a 200 code and “Success” message.

6. edit-letter (PUT request)

    - This endpoint will be a PUT request. It is used to edit letters and save those changes to the database. This operation is only available to admin users.

    - Admin can view the list of letters and select one letter, make their changes, and hit save.

    - From there, we detect if any changes were made to the entry. If so, we call the API and update the appropriate fields. We then return a success code back for the client side to handle.

7. delete-letter (POST request)

    - This endpoint will be a POST request. It is used to delete letters from the database. This operation is only available to admin users.

    - We ask the user on the front end if they are sure if they want to delete the letter and if so, we delete the letter from the database by its ID. If successful, we return a success message for the client side to handle.

8. get-letters (GET request)

- This endpoint will be a GET request. It is used to get all the letters stored in the database. This operation is available to all aspects of the product, such as admin operations and map visualization.

- We fetch all the letters, and they are displayed for the admin to see in a list format.

Map Plotting System

The API endpoints for this will rely on the request we perform when querying for letters. Behind each letter component will lie values to help plot this item such as location, recipients, historical map id, and much more.

1. plot-letters (GET request)

    - This endpoint will be a GET request. It is used to get all the letters and plot the letters appropriately on a historical map.

    - We get all the letters and send back the coordinates along with some other map plotting info for the frontend to handle

2. get-letter-by-id (GET request)

    - This endpoint will be a GET request. It is used to find a letter and its full information by their ID.

    - We can use this to prevent handling all of our data on the client side and save storage for larger scale use.

Querying System

Like our admin system, this will contain one core API call that gets all the details for each letter. Behind each pin on the map, we will contain all the important details to help plot our letters appropriately.

1. search-letters (POST request)

    · This endpoint is a GET request. It will be used to search for anything related to any of the letters when a user performs a search.

    · A user can search by things such as keywords, tags, description, etc. From there, we submit this information to this endpoint, and we then perform a large set of queries where informationSent is LIKE columns1, columns2, etc.

    · After this is performed, we remove any duplicate entries found in our search, and then return this back for the client side to handle. The return object will be an array of JSON objects.

6.4 Database System

When designing the backend, we considered requirements of admin accounts, ways to store/retrieve letters along with map plotting requirements such as coordinates, tags, and other metadata. Below is our entity relationship diagram along with detailed explanations with what each field means in both tables.

6.4.1 Users Table

We will go in depth on which each field means in the Users table, which we will be referencing Figure 6.4.1.A.

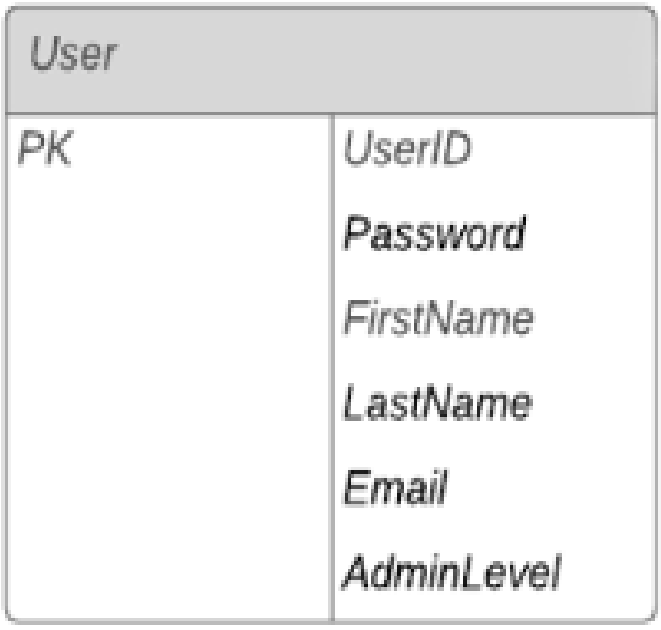


Figure 6.4.1.A. User Entity Description

- UserID
    - Primary key of the admin table that will be auto incremented

- Email
    - Email is used to login to the admin portal
- Password
    - Passwords are stored and encrypted using bcrypt, which using a hashing algorithm that takes numerous years to crack
- AdminLevel
    - Used to determine the power of an admin account. AdminLevel helps determine permissions and is editable only by the super admin.

6.4.2 Letters Table

We will go in depth on which each field means in the Letters table, which we will be referencing Figure 6.4.2.A.

Letters	
LetterID	
	LetterImageID
	LetterTranscript
	Event_des_short
	Event_des_long
	Rollins_letter_num
	Address
	Lat
	Lon
	Owner_1
	Owner_1_loc
	Owner_2
	Owner_2_loc
	Current_owner
	Current_loc
	TimePeriod
	Current_lat
	Current_long
	RecipientName
	RecipientName2
	RecipientName3
	RecipientLocation
	Tags
	Themes

Figure 6.4.2.A. Letters Table

- LetterID
    - Auto incremented primary key
- LetterImageID
    - Images will be stored locally in the repo so the LetterImageID will be used to help retrieve the proper image of the letter

- LetterTranscript
    - Transcribing of the actual letter for those who find the letter hard to read. This also gives the user the ability to query for key words in letters and retrieve a finer result.

- Event_des_short
    - Short description of what the letter contains. This can be a brief sentence summary of the letter's findings
    - This column will be used in our querying when the user is searching for key words/phrases for the letter
- Event_des_long
    - Long description of what the letter contains. This will be more of a summary of the letter and will be a slightly longer/continued version of the short description.
    - Column will be used in our querying when the user is searching for key words/phrases for the letter
- Rollins_letter_num
    - Different reference to the letter number

- Address
    - Address of the letter to where it was being sent.
    - Helps in the map plotting process along with giving the user more details to search by when performing a query
- Lat
    - Latitude of address

- Lon
    - Longitude of address
- Owner_1
    - Owner of the letter. There can be multiple owners since the letter could have been passed on to a different recipient or keeper of the letters.
    - Can be used to display some of the temporal time slider traveling process from owner1 to owner2 to the current owner
- Owner_1_loc
    - Location of the first owner. Formatted as an address of some sort
- Owner_2
    - Owner of the letter. There can be multiple owners since the letter could have been passed on to a different recipient or keeper of the letters.
    - Can be used to display some of the temporal time slider traveling process from owner1 to owner2 to the current owner
- Owner_2_loc
    - Location of the first owner. Formatted as an address of some sort
- Current_owner
    - Current owner of the letter. Letters can be in places such as museums, historical landmarks, and other various sites
- Current_loc
    - Current location of the letter. It will be an address, in more specific
- TimePeriod
    - Time period helps choose the right map for plotting purposes. We'll use historic maps to give the user a better enhancement of the overall product.
- Current_lat
    - Current latitude of the letter. Used for map plotting purposes

- Current_long
    - Current longitude of the letter. Used for map plotting purposes
- RecipientName
    - Name of the recipient of the letter
    - Column cannot be null since there is always a recipient to every letter
- RecipientName2
    - Name of the recipient #2 of the letter. If there are no extra recipients, this column will be null.
- RecipientName3
    - Name of the recipient #3 of the letter. If there are no extra recipients, this column will be null.
- RecipientLocation
    - Location of the recipient of the letter. Latitude and Longitude.
- Tags
    - A tagging system will be implemented where users can search for tag specific entities such as illness, birth, or anything in nature to the letter.
- Themes
    - Themes such as war, poetry, or normal conversations will be marked down to help users in the overall query process.

7 Testing, Prototyping and Evaluation

7.1 Testing

To ensure a properly working project, it is imperative that both the entire project as well as each individual piece is tested. The methods we will use will not only help prevent future issues but streamline the entire development process. We will be implementing unit testing as well as peer reviewed code.

7.2 Unit testing

Unit testing[46] is the process by which one tests and verifies that individual units of code, such as functions, work as expected. The primary reason for unit testing is the speed at which it can be done combined with its effectiveness. Most of the time it is very simple to test a specific unit of code which will very quickly show either success or failure. This quick turn around on results allows the project to be more bug free and feature rich.

7.3 Peer Review

Everyone makes mistakes. To minimize this phenomenon, having our peers review what we have done is imperative. Each section of the project has an overlap of at least two members. For example, the frontend has two developers with a full stack dev working on all aspects of the program. Using the Trello board to track what is currently being done and what has already been finished and GitHubs version control, we can organize who wrote specific code and ensure another member reviews it potentially using unit testing. Not only will this minimize mistakes, but it will also encourage the sharing of ideas and techniques discovered by the team to produce an even better project and speed up the rate at which it was accomplished.

7.4 Prototyping

During our prototyping stage, it was crucial to get some of the barebones functionality working so we can later expand on it throughout this project. Initializing the database and creating the main API endpoints were crucial for the front-end success since they will have to account for handling all that incoming data in a formatted fashion. Creating most of the API routes such as admin CRUD operations and basic querying were essential in early prototyping stages.

7.5 Evaluation

We will be releasing three key evaluation periods:

1. Alpha Testing, where we will have bare-bones functionality of our system and can perform basic CRUD operations through the admin portal
2. Beta Testing, where we will have bare-bones functionality of our map plotting system and give the user the ability to create, edit, and view different historical maps with given plotting data
3. Charlie Testing, where we will deploy our full system and keep track of any additional features, goals, or bugs we encounter during this stage.

8 Other Proposed Solutions

While developing this product, we collaboratively came together on some map solutions such as Google Maps API and proposed that as a viable alternative to Leaflet.js. However, Google Maps API is not very friendly when it comes to time sensitive maps, so we had to roll with Leaflet. In addition, we proposed the idea of hosting this platform on Heroku with a free tier. However, we later discovered the project must be hosted on the UCF CHDR server. This caused some issues initially since we had minimal experience integrating a MERN stack application into a Linux/Apache server.

Overall, we all came up with our own solutions for this project and collaboratively began to see what we envisioned this project to be with our customer. We needed a MERN stack application that could be hosted at UCF and with the help of our sponsor, we were able to successfully gather requirements and begin this project.

8.1 Other Proposed Solutions For Front End

One other solution we talked about to implement the front-end of this project was using PHP with HTML pages and maybe additional JavaScript for functionality. This also basically means that it's going to be another technology stack, perhaps LAMP. Linux, Apache, MySQL, and PHP. It is also a good option since this project requires MySQL as its database. According to the famous survey, 77.5% of the websites use PHP[47]. This of course means PHP is widely used and a great language. One great example of PHP and MySQL is WordPress hosting sites.

Even though this could work and give us the results we want, this feels very outdated. When we have JavaScript options like React, Vue and Angular and their functionality and compactness coming from those libraries and frameworks, it is hard to go against JavaScript. And libraries like Material UI and Ant Design within React are extremely useful and give us a great range of options as well as ease of use as we discussed before.

Another reason to choose newer JavaScript options like we stated above, React, Angular etc., instead of PHP and HTML, is to learn and improve ourselves in those new and upcoming technologies. Computer Science is evolving very fast and new technologies are coming up almost every day. It is up to us and very important to learn and be proficient in those technologies and adapt to change. Therefore we think JavaScript is better for this project and React is an appropriate library to implement this project.

8.2 Our Original Ideas

Below are the original ideas when coming up with our system design. Overall, we collectively agreed about mutual things such as React, MySQL, and more.

8.2.1 Joshua Colston

- MERN Stack with MySQL for the overall application
- Use Heroku for hosting the developmental server and CHDR for the production deployment
- Leaflet.js for the map data visualization aspect of this project. This is also a requirement that was given to us.
- Creating the wireframes in Figma for not only us as developers but a way for customers to visualize what we have envisioned for this product
- Implementation of AGILE scrum workflows using a Trello project board. We will also have our customer in our discord server for any additional questions we may have.
- Create an open sourced API documentation with SwaggerHub so future developers and researchers can use this technology to implement their own John Keat applications
- Research how to use a MERN stack in a Apache server

- Look into an easy yet effective way to upload future letters or poems for admin such as a python script that posts to the api database or reads directly off an excel spreadsheet

8.2.2 Ege Topcuoglu

- Considering starting with the MERN stack with Heroku for ease of implementation then transferring into CHDR server.
- Creating prototypes using Figma or other software so that we can get an idea of what we want and how we can implement it.
- Researching Leaflet.js library as we will implement the map using that
- Researching various React libraries and UI Frameworks to find a easy to implement, easy to use, and clean solution to front-end development of this project
- Research API documentation solutions, considering maybe SwaggerHub as our first option.
- Learn the CHDR server on how to use and deploy on it.

8.2.3 Tyler Goldberg

- MEAN or MERN stack using MySQL for the database work.
- Leaflet is a given requirement for the map visualization using popular plugins such as heatmap.js and leaflet.minicharts.
- Figma seems to be the only application that is both free and has our necessary specifications for a prototyping tool.
- Use Github for our version controlled collaborative code hosting platform.
- Implement the application on the CHDR server as that is a requirement.

8.2.4 Ryan Philbin

- Along with the required heat map, explore other ways to display the data on a map, and allow the user to choose their view.
- Highlight a random “letter of the day”, and on click, show the letters path on a map, and show details about it.
- If time allows near the end, we could make a page that links to other resources to John Keats’ work or more information about who he was.
- In addition to showing what cities/countries were most popular, we might want to provide some context as to why they are so popular. Maybe list where Keats lived, or traveled for work or vacation, etc.
- Using the temporal slider, create a short animation of some letters flying around the map, so when users visit for the first time, it is not just a static page. It would definitely give our website some life.
- Consider expanding the maps to where he wrote his poems, or even build a template to expand this website to other poets, or famous letter series.
- When writing to another well known person (if applicable) link to more info about that person or provide a small paragraph explaining who they are.

8.2.5 Thanah Raveendran

- Consider swapping out to React for frontend development to allow for a more modern and smooth user experience.
- Deploying through Heroku may lead to a much smoother experience.
- Explore more non-admin user utility and use; improve its use as a research tool.
- Since the application will be available to a general audience, how can we improve accessibility tools? We can further explore tools for visually impaired users, such as audio tools, or perhaps even translating tools as a possible functionality to expand on later.

- Considering the potential scale of the documents; there is a finite number of letters to account for, where might further expansion come from?.
- Scalability in general; at its core the customer is looking for a simple database accessing app that includes admin users with CRUD operations, how can this be expanded on to increase use and value.

8.3 Collective Contributions

- Customers mentioned various UI tools for ease of access for users, we can aim to explore various ways to simplify user experience.
- Negotiated slightly different technologies to use with customers to make the web application both appear more modern and run more efficiently.
- Establish an effective foundation to both be released successfully and be expanded upon easily by means of extensive planning
- Looking into implementation of a simple to use letter uploading system, where admin can upload a potential letter with ease.
- Researching React Native and potentially porting this technology to a mobile application as a stretch goal for on the go data visualization.
- Add some interactive video tutorials on how to use the maps, and even open sourcing the API documentation for other users to use our db in their projects.

9 Budgeting and Financing

In Figure 9.A, we came to the conclusion that our product will cost nothing to host and run. We are doing our development on the UCF CHDR server along with using open source tools such as Leaflet. In addition, all of John Keats letters are available to the public. We may look into some graphic design for the website logos, but our sponsor informed us that will be a decision for a later time. They did give the go-ahead that if anything was found that would cost, to bring it to them for approval. If accepted the price would be covered.

Resource	**Costs**
Graphic Design (TBD)	$0
Version Control/GitHub	$0
Deployment Server Hosting w/ CHDR	$0
API Documentation (TBD)	$0
Access to Keats Letters	$0
Developmental Servers (Heroku)	$0
WireFrames	$0
Leaflet.js	$0
Software Licensing	$0

Table 9.A Budget Table

10 Project Management

10.1 Overview

To develop the designs for this document, great organization was essential. Starting with a core foundation of team empowerment and creating the “Circle of Safety” is critical to empowering each group member. Early in development, it was decided Josh Colston would serve as the project manager.

“Only when we feel we are in a 'Circle of Safety' will we pull together as a unified team, better able to survive and thrive regardless of the conditions outside.” – Simon Senek

Providing a project environment that not only made everyone feel empowered, but it also drove critical thinking skills for the overall system design along with the creativity of making a full stack web application.

Initially, this project began with a waterfall approach with the system design, prototyping, and organizing roles/responsibilities. From there, we began to migrate to an agile approach inspired by Scrum. Transitioning from waterfall to scrum gave some initial issues as everyone was adapting to the new workflow, but it seemed to work out in the end.

We tried our standup meetings with the Orli Bot, but it seemed to not be very intuitive in the overall grand scheme of a standup meeting. From there, we began to do our daily scrum meetings just over discord two days a week.

Furthermore, one day a week we began a “working meeting” where we got together and got any additional work that needed to be done such as wireframe discussions, deciding roles, or creating the biweekly sprints using our backlog. Overall, it was a great experience using agile and it seemed to work out in the end.

After finishing our initial design document and gathering requirements from our customer, we began to migrate towards an agile workflow, since a lot of the research for different technologies had begun. We then began our first official sprint and started doing standup meetings.

The following section contains some key artifacts from this entire process, including:

- Product Backlog
- Milestones
- Technologies Used

10.2 Product Backlog

Once we finally embraced the agile methodology, we started to move our attention towards a product backlog. In Figure 10.2.A, we began to prioritize some of the prototyping and getting our API routes setup. It was crucial to have the API routes setup before the end of Senior Design I because that would give the front-end team plenty of time to begin testing some functionality.

Please note that a burndown chart was not presented. Unfortunately, we did not have enough time to create a burndown chart but will be implemented later in Senior Design II.

PRODUCT BACKLOG						
User Story ID	User Story	Estimate (size)	Priority	Sprint	Task owner	Estimated effort
US001	Initialize Repo and Production Serve	Small	5	1	Project Manager	1
US002	Assign Roles and Responsibilities	Medium	5	4	Everyone	5
US003	Create Initial WireFrames for Customer	Large	4	1	Frontend	2.5
US004	Initialize React app in repo	Small	5	2	Project Manager	1
US005	Begin creating an admin portal with barebone functionality	Large	3	4	Full stack/Frontend	6
US006	Load Sample Data into Letters Table	Medium	3	3	Database	1
US007	Complete 15 Pages of overall design document	Large	5	3	Everyone	10
US008	Begin creating API routes to perform barebone querying in regards to map visualization	Medium	3	4	API/Full stack	3
US009	Begin to experiment and prototype with historic maps for the UI	Medium	3	4	Front End	10

Table 10.2.A Product Backlog Table

10.3 Project Milestones

During our initial design document, we came up with these general milestones throughout our project. By the end of the first semester, we should have our design document completed along with a majority of the API routes added.

1. Communications (2/2/22)
2. Role allocations (2/7/22)
3. Project scopes requirement + customer approval agreement (2/7/22)
4. ERD (2/13/22)
5. Gantt Chart (2/15/22)
6. Block Diagram (2/14/22)
7. Database Model (2/21/22)
8. Technology research (2/28/22)
9. Data uploaded into DB (3/1/22)
10. CRD w/ Leinecker (3/2/22)
11. UI Layout WireFrames (3/6/22)
12. Second TA check-in: week of 4/4-4/11. 20-25pg Powerpoint
13. Documentations (general; add specifics to this i.e. API, project documentation, etc) (4/27/22)
14. Prototyping/ testing components (4/27/22)
15. Final Design Document Due: (4/27/22)
16. API Completed (4/27/22)
17. Front End Functional (9/1/22)
18. UI/UX Completed (11/30/22)
19. Feature Completeness (11/30/22)
20. Initialized Production and Developmental Servers

10.4 Technologies Used

Furthermore, we used a various set of communication tools and project management libraries to help keep us on track throughout this semester. In this section, we will talk about those tools.

Trello

Trello[48] is a powerful collaboration and project management tool that assists with team productivity. It enables us to create and update the system on the fly for whatever our current needs are. We are currently utilizing seven individual sections to organize our workflow as shown in Figure 10.4.A:

1. **Project Resources**

Project resources is where we keep access to all required documents, GitHub, and links needed for both the research section of the project and during actual implementation. This includes the link to our group Google Drive and ongoing sponsor zoom link, as well as GitHub repo.

2. **Upcoming Dates**

Upcoming dates state the next incoming assignment due and their date for things such as needing fifteen individual pages done or the CRD. This ensures individual responsibility for being prepared.

3. **Next Meeting**

Next meeting is the place where we note down both what we said we would get done before the next sponsor meeting, and where we mark down questions or topics to bring up.

4. **To Do**

To do is exactly what it sounds like, a list both broad and specific with tasks that need to get done. With Trello, members of the group can tag posts indicating that they will handle that specific task. Opening a task will bring up more details about the indicated assignment.

5. Pending

Once a member selects a task they will handle and begins working on it, they move that task to Pending. This indicates to other members productivity for this task.

6. Blocked

Assignments in the blocked category are ones that cannot be completed until another task has been done. The assignment will clearly state what needs to be done and who, if anyone, is currently working on it.

7. Done

Cards in the Done section indicate assignments, dates, and meetings that required prep work that have been completed. They will also include any important notes or suggestions that were brought up during such times.

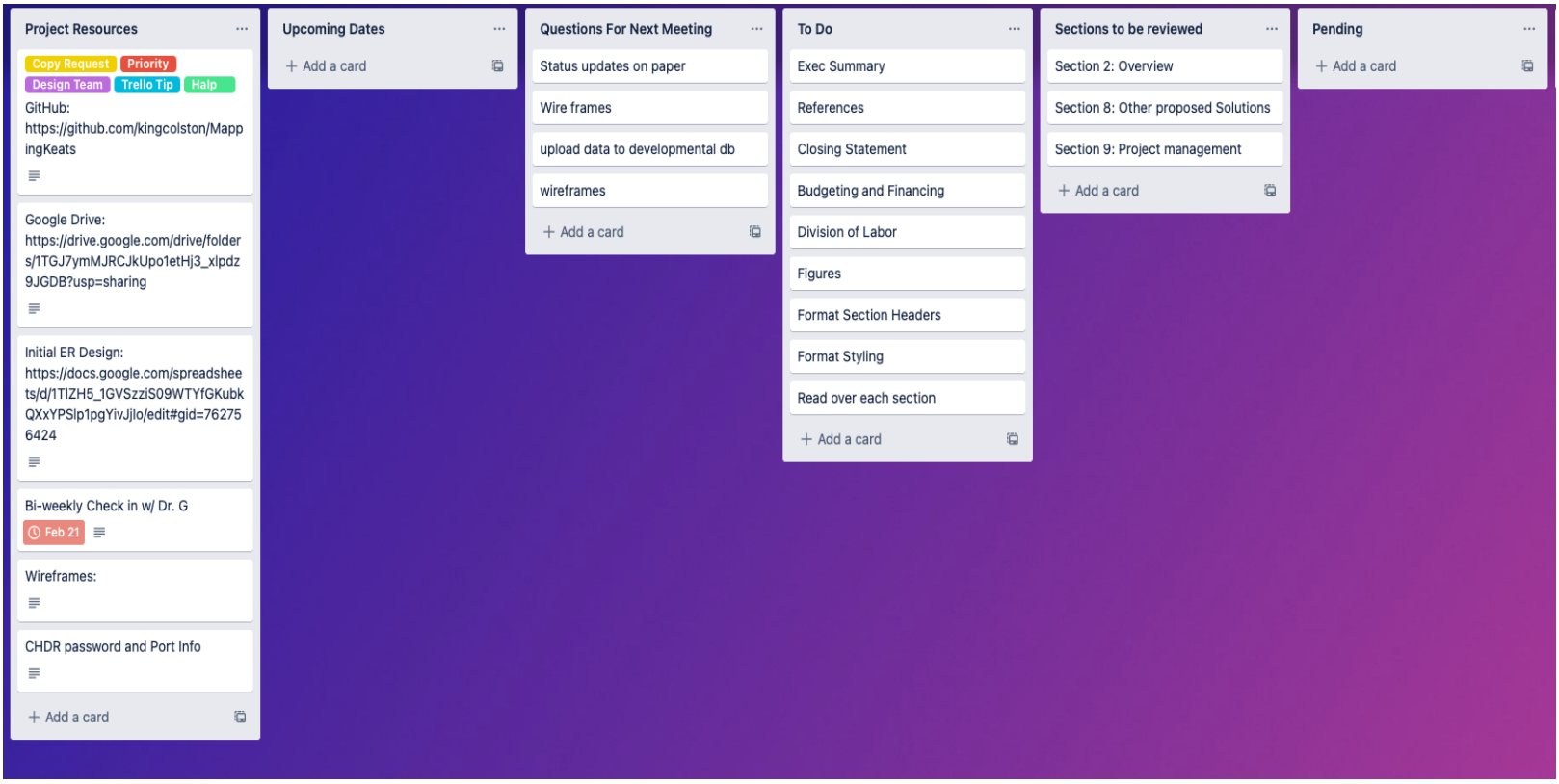


Table 10.4.A Trello Board Organization

Discord

Where Trello is our official tool for future planning and organization, Discord[49] is our application of choice for constant communication and ideas. It ensures every member is kept apprised of the goings on of other members and if anyone ever has a problem, it is first handled there. When the team is not meeting in person or with our sponsors, we gather in the Discord. This is also an area where our sponsor can get quick word to us and monitor progress beyond emails and our meetings every other week. Our Discord is organized into a few main categories as shown in Figure 10.4.B:

- **General**
    - General is where all casual conversation takes place between members. Whether its other school business or simply slightly off-topic from the project, it all fits in here.

- **Frontend**
    - This channel is for the two front end developers and the full stack member to communicate ideas and discussions, so they don't get mixed up with the other sections. Resources specific to the frontend including wireframes are also posted here.

- **Backend**
    - This channel is for the backend developer and the full stack member to communicate ideas and discussions, so they don't get mixed up with the other sections. Resources specific to the backend including an ERD can also be posted here.

- **API**
    - This channel is for the api developer and the full stack member to communicate ideas and discussions, so they don't get mixed up with the other sections.

- **Availability**

    - We all have lives outside of this project and things are always coming up. This is where members post about upcoming dates and times that they cannot make due to other obligations.

- **Meeting Notes**

    - During every sponsor meeting, the group collects notes on what was said and asked for and places them here for future viewing (and sorted for specific requirements for the Trello board).

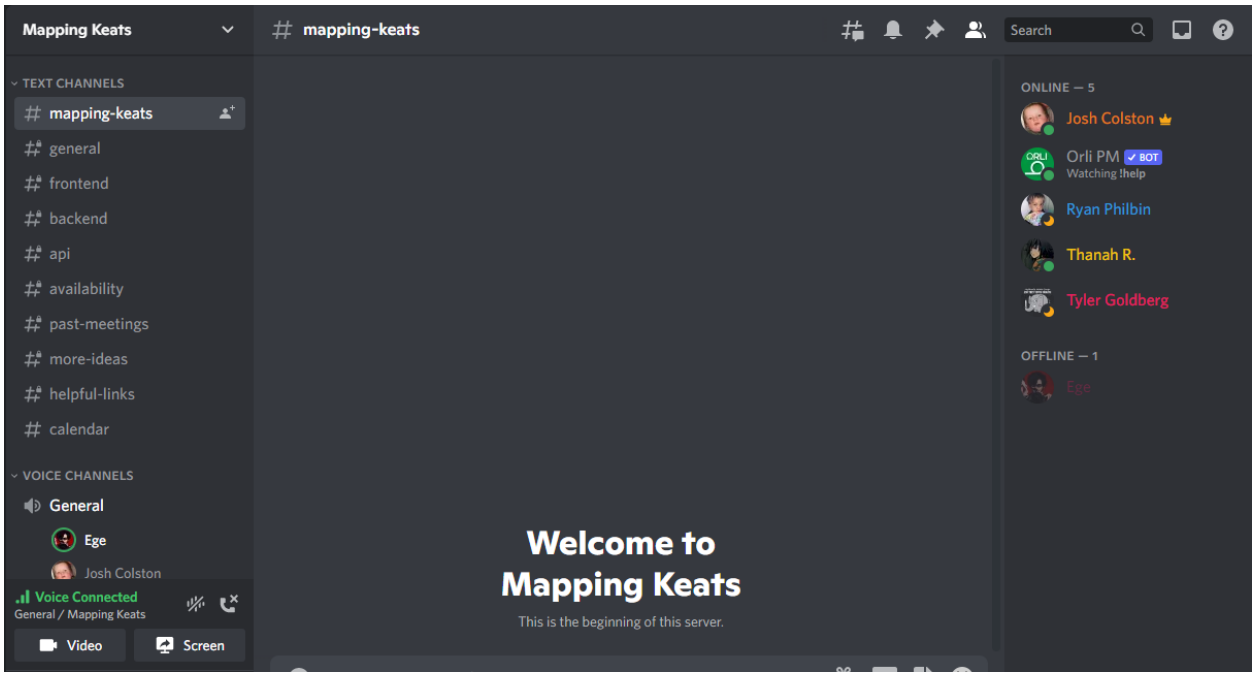


Table 10.4.B Discord Organization

- **Zoom**

    - While Zoom[50] isn't used very often, it is the primary form of contact with the sponsors. As part of our sponsor team is located out of state and most people are familiar with Zoom after the recent pandemic, this is the tool used at least every other week for "face-to-face" meetings.

- **Email**
    - When resources need to be passed between the project team and the sponsor team, it is done through email. This allows us to share data such as an example database schema as well as example data that needs to be stored in the database.

11 Work for the Future

When designing the overall product, we considered future ideas that can be later implemented for either a new Senior Design group or whoever chooses to maintain the project overtime.

- **Make the Application Generic**
    - Admins can add more authors to this web application and expand it to more than just John Keat. For example, if the admin wanted to track letters of Ben Franklin and give users the ability to perform the same operations. This would provide a more generic application use and can give students/researchers to visualize anyone's letters on a single web application.

- **Track user Analytics**
    - Tracking user analytics would give the developers an insight on where/how users interact with this application to not only improve UI changes but add additional functionality.

- **User accounts**
    - Adding Accounts for users would be a great way for frequent users to save their queries, map searches, and filtering with one touch of a button.

Overall, we believe these ideas can be implemented for a long-term solution to research not only just John Keat, but other authors that had traveling letters/poems.

12 Summary and Conclusion

If we look back to this design document and summarize the process, our goal is to build a website that is going to store John Keats' letters and show them in a way that is simple, interactive, and useful. This website will be an educational website that will show John Keats' work and it will put a light on the lifestyle at that time John Keats lived. With the amount of data we have from the letters, we will be able take a look into that timeline and we will be able to take that data and put it in front of the users by showing the picture of letters itself, a transcript of the letter, original location, recipient location and some much more on a map with different toggles such as heatmap or historical map.

Our website offers a combination of systems such as, an Admin system, API System, Querying System and Map Plotting System as we discussed in our document. Admin System is for our sponsors in which they can add, edit, delete John Keats letters. The API system is there for all functionality from database queries, to uploading files to the database. The Querying System is for searching capabilities for both Admins and Users. And Map Plotting System is for visualization of our data in a map consisting of different toggles.

To achieve all of our goals and requirements, all of our group members have researched various technologies, libraries and frameworks to choose what fits in this project while counting on the requirements and constraints we have. With all the knowledge and experience we have acquired both from Senior Design 1 and our previous classes we have compiled this document and we have created the roadmap for our project for Senior Design 2.

In conclusion, we are proud of the work we put on this project so far, and we are proud to present this document. Our communication between group members is extremely good and every group member is eager to learn and improve, which makes a great environment in our team meetings. We are looking forward to Senior Design 2, where we can use our research and experience to implement what we have designed, hit all of our stretch goals, and make a great end product that we can talk about in the future of our careers.

References

[1]University of Central Florida. (n.d.). Center for Humanities and Digital Research - CHDR @ UCF. Retrieved April 25, 2022, from https://chdr.cah.ucf.edu/

[2] Hough, G. G. (2022, March 7). *John Keats, British Poet.* Britannica. Retrieved April 25 ,2022, from https://www.britannica.com/biography/John-Keats

[3]McCarthy, A., Newman, I., Rejack, B., Singer, K., Stanback, E. B., & Theune, M. (2022, January 24). The Keats Letters Project. Retrieved April 25, 2022, from http://keatslettersproject.com/

[4]Wikimedia Foundation. (n.d.). *React (JavaScript library)*. Wikipedia. From https://en.wikipedia.org/wiki/React_(JavaScript_library).

[5]Kim, M. (2021, December 9). *Declarative vs Imperative - Myung Kim*. Medium. From https://medium.com/@myung.kim287/declarative-vs-imperative-251ce99c6c44

[6]*Components and Props* –. (n.d.). React. From https://reactjs.org/docs/components-and-props.html

[7]*Pros and Cons of ReactJS - javatpoint*. (n.d.). www.Javatpoint.Com. From https://www.javatpoint.com/pros-and-cons-of-react

[8]*React JSX*. (n.d.). JSX Examples. From https://www.w3schools.com/react/react_jsx.asp#:%7E:text=JSX%20allows%20us%20to%20write%20HTML%20elements%20in%20JavaScript%20and,easier%20to%20write%20React%20applications.Romanyuk, O. (2020, March 19).

[9]*Angular vs React: Which One to Choose for Your App*. freeCodeCamp.Org. From https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/ E. (2021, November 10).

[10]*Vue.js vs React: Comparison of Two Most Popular JS Frameworks*. Codica - Custom Software Development Consultancy. https://www.codica.com/blog/react-vs-vue/#:%7E:text=When%20it%20comes%20to%20the,HTML%20templates%20apart%20from%20JSX.

[11]freeCodeCamp.org. (2018, April 15). *Meet Material-UI — your new favorite user interface library*. From https://www.freecodecamp.org/news/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c/

[12]*React-Bootstrap*. (n.d.). React Bootstrap. From https://react-bootstrap.github.io/

[13]*Introduction - Ant Design*. (n.d.). Ant Design. From https://ant.design/docs/spec/introduce

[14]*Home - Fluent UI*. (n.d.). Fluent-UI. From https://developer.microsoft.com/en-us/fluentui#/

[15]R. (n.d.). *GitHub - remix-run/react-router: Declarative routing for React*. GitHub. https://github.com/remix-run/react-router

[16]Kelhini, F. (2022, February 8). *Axios vs. fetch(): Which is best for making HTTP requests?* LogRocket Blog. Retrieved April 26, 2022, from https://blog.logrocket.com/axios-vs-fetch-best-http-requests/

[17]*An open-source JavaScript library for interactive maps*. Leaflet. (n.d.). Retrieved April 25, 2022, from https://leafletjs.com/

[18]*Latest*. OpenLayers. (n.d.). Retrieved April 25, 2022, from https://openlayers.org/

[19]*Leaflet vs OpenLayers. pros and cons of both libraries*. Geoapify. (2020, November 5). Retrieved April 26, 2022, from https://www.geoapify.com/leaflet-vs-openlayers

[20]*Maps, geocoding, and Navigation Apis & sdks*. Mapbox. (n.d.). Retrieved April 25, 2022, from https://www.mapbox.com/

[21]Temprano, V. G. (n.d.). *Pros and cons of using Mapbox for your project*. Codementor. Retrieved April 26, 2022, from https://www.codementor.io/@victorgerardtemprano/pros-and-cons-of-using-mapbox-for-your-project-dx04pfgxw

[22]Google. (n.d.). Google maps platform | google developers. Retrieved April 25, 2022, from https://developers.google.com/maps

[23]Temprano, V. G. (n.d.). *Google maps API or leaflet: What's best for your project?* Codementor. Retrieved April 26, 2022, from

https://www.codementor.io/@victorgerardtemprano/google-maps-api-or-leaflet--what-s-best-for-your-project-faaev60vm

[24]*Map Warper*. Home - Map Warper. (n.d.). Retrieved April 25, 2022, from https://mapwarper.net/

[25]*Node.js Introduction*. Node.js introduction. (n.d.) From https://www.w3schools.com/nodejs/nodejs_intro.asp

[26]*The advantages and disadvantages of Node.js Web App Development*. The Advantages and Disadvantages of Node.js Web App Development. (2022, January 11). From https://www.mindinventory.com/blog/pros-and-cons-of-node-js-web-app-development/

[27]TutorialsPoint. (n.d.). ExpressJS. From https://www.tutorialspoint.com/expressjs/index.htm

[28] *(n.d.) What is Express.js?* Besant Technologies. Retrieved April 25,2022, from https://www.besanttechnologies.com/what-is-expressjs#:~:text=It%20is%20used%20for%20designing,and%20API%20without%20any%20effort.

[29]*Lang.github*. elixir. (n.d.). From https://elixir-lang.org/

[30]*PERL*. Perl - introduction. (n.d.). From https://www.tutorialspoint.com/perl/perl_introduction.htm

[31]*Asp and ASP.NET tutorials*. ASP Tutorial. (n.d.). From https://www.w3schools.com/asp/

[32]Wikimedia Foundation. (2022, April 17). *Object–relational mapping*. Wikipedia. From https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping

[33]Bodnar, J. (n.d.). *Sequelize tutorial*. Sequelize tutorial - JavaScript ORM programming with Sequelize. From https://zetcode.com/javascript/sequelize/

[34]Pattinson, T. (2021, October 19). *Relational vs non-relational databases*. Relational vs Non-Relational Databases. From https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases

[35]*What is a non-relational database?* MongoDB. (n.d.). From https://www.mongodb.com/databases/non-relational

[36]*MySQL 8.0 Reference Manual :: 1.2.1 what is mysql?* MySQL. (n.d.). From https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[37]Contributor. (2020, March 2). *8 advantages of using mysql*. DevOps.com. From https://devops.com/8-advantages-using-mysql/

[38]*What are the components of the mysql server?* Educative. (n.d.). From https://www.educative.io/edpresso/what-are-the-components-of-the-mysql-server

[39]Wikimedia Foundation. (2022, April 9). *PostgreSQL*. Wikipedia. From https://en.wikipedia.org/wiki/PostgreSQL

[40]Aggregation, A. P. on D., Integration, D. D. S. on D., & Integration, O. J. on D. (2022, April 6). *PostgreSQL vs MySQL: 8 critical differences*. Hevo. From https://hevodata.com/learn/postgresql-vs-mysql/

[41]*MySQL vs. mongodb: What's the difference?* IBM. (n.d.). From https://www.ibm.com/cloud/blog/mysql-vs-mongodb

[42]*The Collaborative Interface Design Tool.* Figma. (n.d.). Retrieved April 25, 2022, from https://www.figma.com/

[43] Capacci, M. (n.d.) *Leveraging Figma's Features for the Entire Design Process*. Toptal. Retrieved April 25, 2022, from https://www.toptal.com/designers/figma/figma-features

[44] Wied, P. (n.d.) *Leaflet Heatmap Layer Plugin*. Patrick-wied.at. Retrieved April 25, 2022, from https://makeshiftinsights.com/blog/heatmaps-leaflet-heatmap-js/

[45] PPete. (2022, March 6). *Leaflet.PolylineMeasure*. Github. Retrieved April 25, 2022, from https://github.com/ppete2/Leaflet.PolylineMeasure

[46]Foo, H. (2020, October 7). *Unit tests and code reviews: The bedrock of software quality*. Medium. Retrieved April 26, 2022, from https://blog.devgenius.io/unit-tests-and-code-reviews-the-bedrock-of-software-quality-9a23cd24558b

[47]*Usage Statistics and Market Share of PHP for Websites, April 2022*. (n.d.). From https://w3techs.com/technologies/details/pl-php

[48]Trello. (n.d.). Retrieved April 25, 2022, from https://trello.com/en-US

[49] *Your place to talk and hang out*. Discord. (n.d.). Retrieved April 25, 2022, from https://discord.com/

[50] *Video conferencing, cloud phone, webinars, chat, virtual events: Zoom*. Zoom Video Communications. (n.d.). Retrieved April 25, 2022, from https://zoom.us/