NM Forest Industry Map and Data Collection Website
Design and Implementation Report

Tri Blankley, Jena Peterson
Contributors: Dominic DOnofrio

May 13th, 2025

Contents

1 Project Overview

This project consists of two parts. First off, the creation of a Landowner database and an Industry database. Next, the creation of a web application that interacts with those databases to facilitate communication between landowners and businesses. This projects utilizes as many open-source resources as possible for the implementation of this project. The only paid service is done through Microsoft Azure for the use of a SQL database. This project is the beginning of a long-term relationship between NM State Forestry division, New Mexico Tech, and Navajo Tech. With this in mind, the team is made this web-app as a starting point for future development for upcoming semesters. The goal is to make this project reusable and easy to build upon, whether that be the databases or the web application.

1.1 Scope & Objectives

- **Current Scope:**
  The scope of this project focuses on the creation of the databases and the web application. First off, the main goal was to create two databases; landowner and industry. Next, the creation of the web application. The web app has a map that has pins of all nearby businesses, ways to filter what businesses are shown (Type of work, location), contains contact information for businesses, and has survey pages for users to submit their information to the databases.

- **Changes in Scope:**
  There was additional planned functionality that was removed for a variety of reasons. Originally, there were plans to let landowners send emails to businesses through the app, to allow businesses to contact landowners, in-site chat to connect landowners and industry, and account creation. These were not included due to time constraints, the complexity of the addition, and to keep this as user friendly as possible.

  However, more focus was put onto the filter functionality; allowing landowners to search for businesses directly and enabling each business to decide on the radius.

  Changes have also been made to the database designs to accommodate changes to the survey questions. For example, the additions of if wildfire or flooding happens on a Landowner's property. However, other questions have been taken out due to how open-ended and vague the answers could be. For example, what type of natural resources does a parcel of land have? Some landowners may not know and that type of information would be in a Forest Management Plan that landowners can submit to the landowner database.

1.2 Requirements

1.2.1 Functional Requirements

The functional requirements of this project are the creation of two databases that house information on landowners and forestry businesses, and the creation of a website that makes communication between landowners and businesses easier. The main idea is that landowners will use this website rather than contacting NM Forestry.

1.2.2 Non-functional requirements

The system was designed with several key quality attributes in mind. Usability was prioritized through a simple and easy-to-use user interface that works seamlessly on both mobile and desktop devices. Security is ensured through the protection of personal information, with Azure providing access control, threat detection, and encryption. Reusability was considered by building the databases and website to support future expansion. Extensibility allows for additional functionality to be added with minimal effort. The system also emphasizes performance, with fast handling of requests and queries, and scalability, using Vue.js and modular components to accommodate future growth.

1.2.3 Stakeholder Needs

- Users:
  - Landowners: Input their information, search for businesses, contact businesses
  - Businesses: Input their information
- Admins:
  - NM Forestry: Input collected data, contact landowners/businesses, maintain databases
- Regulatory Bodies:
  - State and Federal Government: Ensure privacy laws are met

1.3 Project Goals

- Usability: Works well on mobile and desktop, simple navigation
- Security: Protect private information in databases
- Reusability: Databases reusable for other projects; web app expandable
- System performance: Fast requests/queries, minimal database queries

1.4 Constraints

- Timeline: This project needs to be completed within the next month.
- Budget: Microsoft Azure gives students credit to use for their servers.
- Compliance: Privacy laws and other data protection standards.
- User license: Using open-source options.

1.5 Use Case

1.5.1 Use Case Scenarios

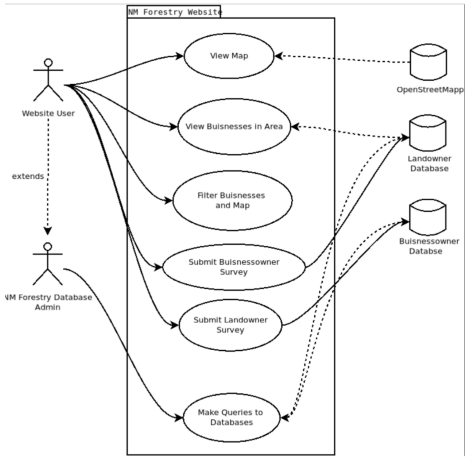

Figure 1: Use-Case of actions accessible to website user. Both private landowners, and industry leaders interface with the same website to enter a survey, individual accounts are not needed

1.5.2 Scenario: Find a business

1. User opens the web application
2. User inputs their location (map shows nearby businesses)
3. User filters businesses based on job type or distance
4. Website shows businesses that fit the criteria
5. User selects a business (contact details shown) and contacts them directly

2 System Requirements

2.1 Required Tools & Resources

- Axios - HTTP client for Node.js-Vue.js communication. Axios is a promise-based api, making it more reactive than callback structured alternatives.
- Docker - Package applications and their dependencies into lightweight, portable containers that run consistently across systems. Also simplified development and deployment, as the developer only needs to run a single command to spin up an instance of website.
- Leaflet - Interactive mapping framework for Vue.js. this Package handles all communication with OpenStreetMap, pin rendering, and map styling.
- Leaflet Animated Search Box - Location search for the map
- Microsoft Azure - Hosting the MySQL databases. Azure systems also provide analytics and security solutions for the databases it hosts.
- MySQL - Relational database system, storing survey response data from both industry leaders and private landowners.
- Node.js - Connects to mySQL databases hosted on Azure Cloud Systems to execute queries and posts. Also provides a wide variety of packages used during website creation.
- Nominatim - Geocoding API used for converting business locations given as street addresses to the longitude-latitude coordinates required to render pins on the leaflet map.
- OpenStreetMap - Open-source alternative to Google Maps, provides map image segments and coordinate mapping.
- Visual Studio Code - Primary IDE chosen for its wide variety of extensions.
- Vue.js - Open-Source Front-end framework tuned to component-based Single Page Applications. This web-engine provides many tools used to simplify the development process by simplifying the process of allowing inter-component communication.

3 Implementation & Design

The project adheres to the Model-View-Controller (MVC) architectural pattern. In this implementation, Vue.js serves as both the view and the controller, managing user input and dynamically updating the interface based on user interactions. Vue.js communicates with the model, comprising the Node.js server and Leaflet, using Axios to send and receive data. This structure enables a clean separation of concerns, allowing for scalable and maintainable development.

3.1 Methodology

The website is be simple; as this project will be given to another development group following the work done this semester. Keeping the project simple will ease future development and additions, reduce the number of security vulnerabilities, and make a more simple and usable website for end users.

The front-end aspect is built in a modern web engine, improving the experience of running and maintaining the website, as modern web engines will receive quality of life and security updates for at least 5 years. Modern web engines also benefit from being flexible, object-oriented systems, which support a smaller, easier-to-read code base for future development. Vue.js is open-source, and has almost complete compatibility with node.js packages. Making it very flexible and easy to work with.

As this project will see continuous development over the next several semesters by future students, the software and datasets are under open-source license agreements to remove limitations to the same tools. The only non open-source tool is the Microsoft Azure MySQL server. The use of one is free for students with a student account on Azure. However, if future students do not want to use Microsoft Azure, they can use the given SQL code to make the databases somewhere else.

3.2 Docker and Development Environment

Dockerizing the Project

To ensure deployment of the website is straightforward and maintainable the website is deployed via a docker environment. Using a multi-container approach to streamline development and ensure consistent machine environments. This allowed us to encapsulate all dependencies, scripts, and server behaviors into isolated environments, making the application easier to test and deploy.

Dockerfile Overview

The root of containerization begins with the Dockerfile, which sets up the base environment for the project. The key components are:

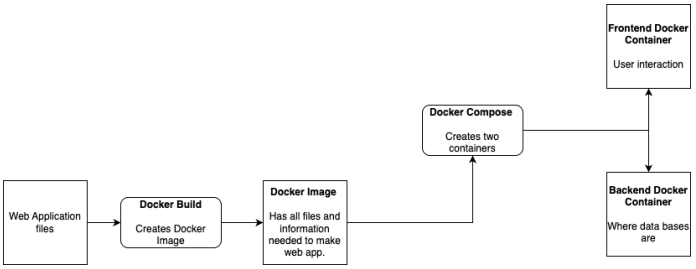

Figure 2: Diagram of Docker setup, showing the two containers the dockerfile builds into.

- **Base Image:** node version `node:18` official image, Was chosen ensuring

compatibility with the Node.js version required by the project.

- **Working Directory:** The container sets `/app` as its working directory using `WORKDIR /app`, where all operations and installations are run.
- **Dependencies Installation:** First, only the `package.json` and `package-lock.json` are copied to the container, and `npm install` is run. This optimizes the build process by caching dependencies.
- **Application Files:** After installing dependencies, the remaining files are copied using `COPY`.
- **Default Command:** The container runs `npm run dev` by default, launching the development server.

This configuration separates dependency installation from application copying, reducing unnecessary rebuilds when only code changes are made.

docker-compose Setup

A `docker-compose.yml` file was created to orchestrate two primary services, as both vue.js, and node.js must be running to access the website.

Primary Services: `dev` and `server`.

- **dev Service**
  This is the main development container. It:
  - Is built from the local Dockerfile.
  - Maps port 5173 on the host to 5173 in the container (common for Vite or similar dev servers).
  - Mounts the local project directory and preserves `node_modules` to allow live reload and local edits without container restarts.
  - Sets the `NODE_ENV` to `development`.
  - Executes `npm run dev`, starting the local development server.
- **server Service**
  This container simulates the back-end server for testing purposes:
  - It also builds from the local Dockerfile.
  - Maps port 3000 from the container to the host.
  - Uses the same volume setup for file persistence and local development.
  - Runs the `server.js` script using the `node` command.
  - Shares the same development environment setup.

Benefits and Outcomes

Using Docker and Docker Compose ensured the front-end and test server could run concurrently without conflict. This approach:

- Eliminated “works on my machine” issues by standardizing environments.
- Simplified onboarding for new developers.
- Allowed parallel development of front-end and back-end modules.

3.3 Deployment model

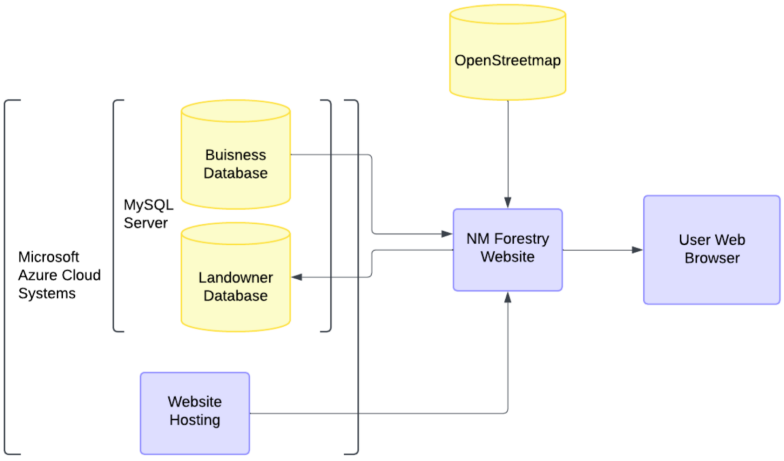

Figure 3: Deployment Diagram of website implementation. Databases and website hosting can be implemented via Microsoft Azure Cloud Systems

- Microsoft Azure Cloud Systems for hosting and databases
- MySQL Server contains the response data for Business and Landowner surveys.
- OpenStreetMap provide map image segments, and coordinate data used to render pins
- NM Forestry Website sends and receives data from these sources to act as the primary interface for users.

Microsoft Azure cloud systems contain all the necessary tools and capabilities to deploy this project in a single virtual machine. Simplifying the cyber-security into a single stack, as well as reducing maintenance requirements. Azure provides industry-standard tools and is already integrated with NM infrastructure. This preexisting integration will reduce the cost of running the website.

3.4 Subsystems

3.4.1 Subsystem Architecture

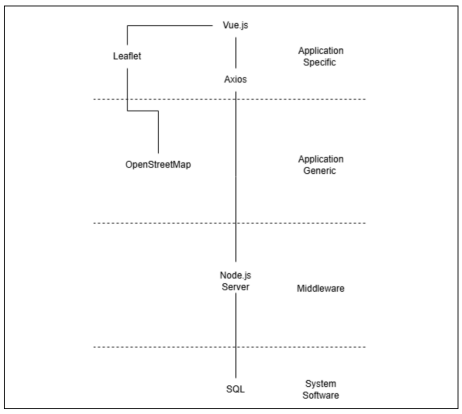

Figure 4: Subsystem Architecture

The subsystem dependency diagram above shows the interactions between all of the components within the subsystems and how it relies on each other; SQL provides the business information, and receives survey information, the Node.js server handles the communication between the databases, and Axios makes sure that Vue.js receives clean JSON data posted by the Node.js server.

Leaflet communicates with OpenStreetMap so that Vue.js can have an interactive map.

3.4.2 Subsystem Design

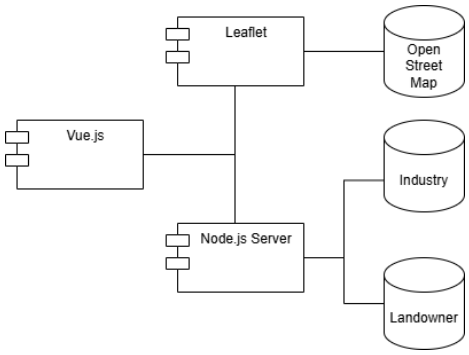

Figure 5: Subsystem Design

There are three important subsystems within the project; the user interface, the map controller via Leaflet, and the databases via Node.js. The UI will be the

primary means by which information will be supplied to the user. When a user accesses the website, the map displayed is rendered from the Leaflet Subsystem, which gets the PNGs of individual map tiles directly from OpenStreetMap. Leaflet will also render pins and information directly on the map view. This data is provided by the Node.js server from the Industry Database. When a user accesses the survey pages, the data they submit will primarily be interfacing with the Node.js server. Which handles all relevant posts.

3.5 Sequence Diagram

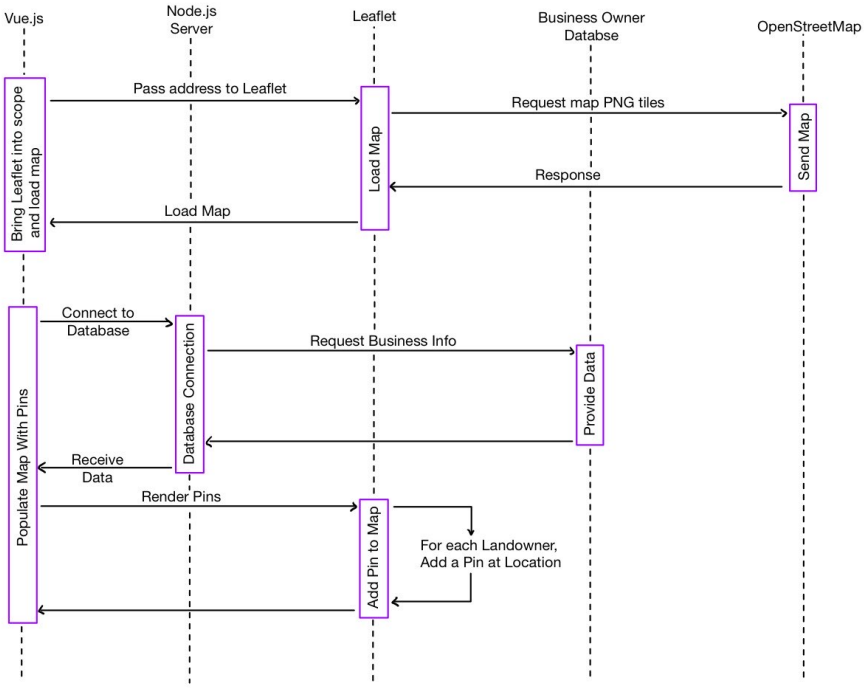

Figure 6: Vue.js receiving data from the databases.

The figure above shows the how Vue.js interacts with Leaflet and the Node.js server. Leaflet receives mapping data from OpenStreetMap and displays it. Vue.js then receives business information from the Node.js server, and that information is sent to Leaflet to put pins on the map.

3.6 Databases

SQL was chosen for its relational capabilities. MongoDB was considered as an alternative but SQL was more suitable. It was decided to use an Azure

MySQL Server as Azure is already in use in cities such as Albuquerque and the MySQL database is free to students.

OpenStreetMap was chosen for being open-source alternative to Google Maps.

- OpenStreetMap: Accessed through LeafLet API
- Industry and Landowner Databases:
  - Hosted on Microsoft Azure SQL server
  - Separate databases that do not interact

3.6.1 Landowner Database

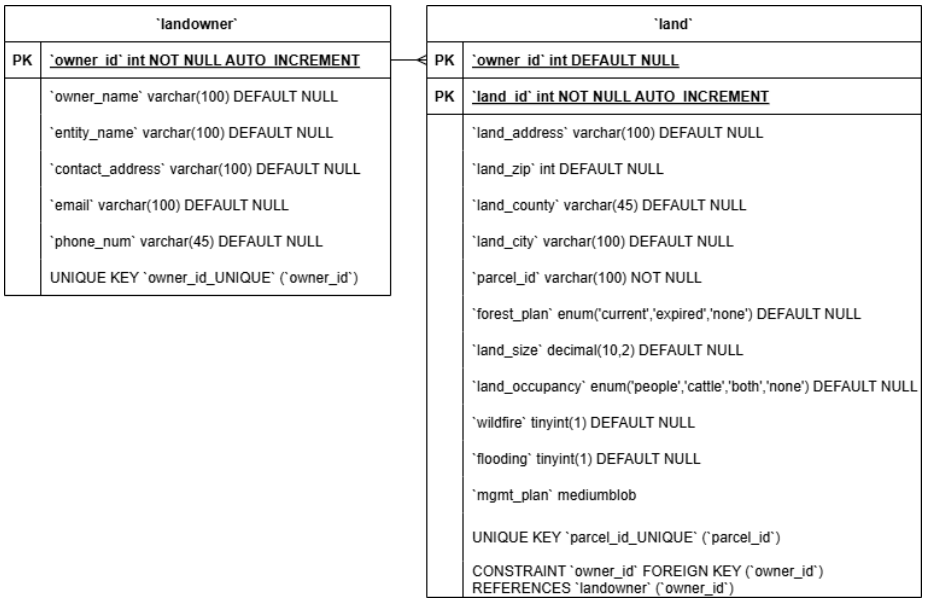

Figure 7: Physical ER Diagram of the Landowner Database.

The figure above shows the setup for the Landowner Database along with more specific information for the columns. There is a one to many relationship between Landowner and Land, as a Landowner could have different parcels of land. This is shown by the owner_id and land_id being Primary Keys for the land table.

The majority of the columns can be NULL if needed, as not all the information is required. It depends on the type of information that is collected.

The majority of columns are varchar due to the possible variety of sizes for the information. However, enum is used for forest_plan land_occupancy to ensure only given answers are inserted. One of the columns, mgmt_plan stores

objects as a MEDIUMBLOB, which is where Forest Management Plans PDFs are currently stored. This is a short-term solution for accepting PDFs from users and should not be used in the long-term due to the amount of storage it takes. MEDIUMBLOB was chosen as it can contain up to 16MB and Forest Management Plans can significantly vary in length.

3.6.2 Industry Database

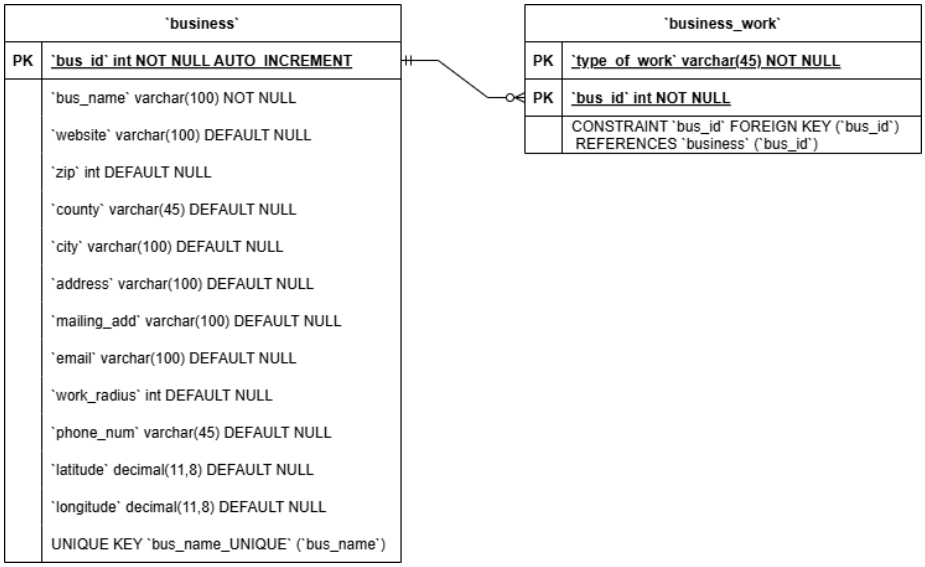

Figure 8: Physical ER Diagram of the Industry/Business Database.

The figure above shows the information for the Industry Database, including the one to optional many relationship between Business and Business_work. As a Business can decide to list the type of work they do but that is not required information.

The type_of_work options are decided by a list contained within the file business_information.ts. Within the application, the type_of_work options are shown as checkboxes, which means users are confined to the options given. This was chosen over an open answer box due to the sheer amount of variety would make it challenging to use it as filter criteria on the application.

3.6.3 Database Testing

Manual testing was done to verify the integrity and functionality of the MySQL datbase. This included confirming that data was correctly stored, table relationships were correct, and that queries returned information in the expected format. This was done through CRUD testing and verification testing to ensure

that unique, null, and auto_increment and other unique values were being handled correctly when creating, updating, and deleting rows. For instance, one test involved retrieving a stored MEDIUMBLOB and converting it back into a PDF file, ensuring that the PDF wasn't corrupted in the process.

3.7 Structure

3.7.1 Website Class Diagram

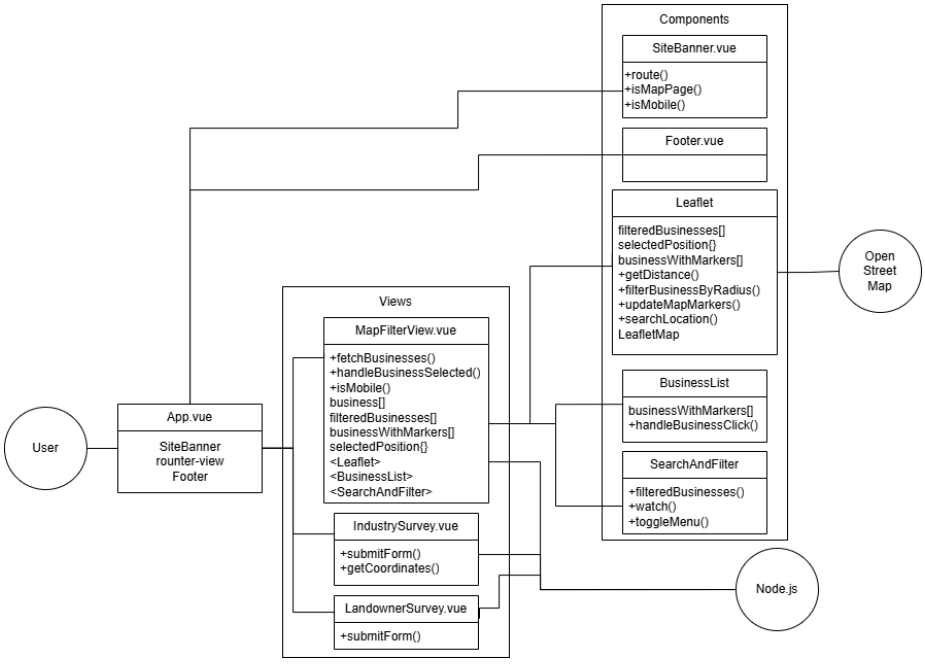

Figure 9: Class Diagram

The class diagram above shows how the main pieces of the website interact and connect with each other. How everything is housed will be explained in the Website Layout.

When the website is first opened, App.vue has the Footer.vue, Header.vue, and MapFilterView.vue on display. The map, list of businesses, and filters will be the first thing a user sees. The MapFilterView contains Leaflet.vue (the map), BusinessList.vue, and SearchAndFilter.vue.

MapFilterView.vue receives the businesses from a JSON that it posted by the server.js through the use of Axios. If it is unable to get these businesses, an error message will be displayed. Any information passed between SearchAndFilter, Leaflet.vue, and BusinessList.vue happen in MapFilterView.vue.

A user can use the filters in SearchAndFilter.vue to find a business. Based on the changes in SearchAndFilter.vue, the business array will be filtered. This

filteredBusinesses is then set to the Leaflet.vue, so only businesses fitting the criteria will appear on the map. Leaflet.vue communciates with OpenStreetMap to have an interactive map and to use coordinates from filteredBusinesses to place markers on the map.

If a user searches a location using the search bar in Leaflet, the search is sent to Nominatim to get the longitude and latitude coordinates. The map will then take the user to those coordinates and show any businesses that have that location within their work radius. These businesses are stored in businessWithMarkers and then sent to BusinessList, which lists every business that matches the specified filters and given search location. If no businesses match the given criteria, BusinessList will ask the user to change their search criteria.

The survey pages are the same in terms of layout but there are differing questions on each. The focus for the LandownerSurvey.vue is to ensure that the user knows that when entering their information that there is a difference between contact information and land information; since someone may not live on a parcel of land that they own and must be contacted at a different address or through an alternative way (phone, email). The survey also includes specific information about the land, such as land occupancy (by people or livestock), and a history of wildfires or flooding. A user is also able to submit a PDF that will be stored as a MEDIUMBLOB within the database.

The IndustrySurvey.vue also requires the use of Nominatim to get coordinates for businesses. When filling out the survey and the user submits it, the address details is sent to Nominatim. If the returned coordinates do not fall within New Mexico or is null, the survey will not submit and the user will be prompted to retype the address. If the business does not have a set address, a city in which the business works is acceptable. A business must have longitude and latitude coordinates to show up in BusinessList.

Both survey pages have required fields that must be filled out. Anything that is left blank or not filled out correctly will stop the user from submitting. Examples of this are incorrectly typed email addresses or phone numbers that do not match the given format. However, the phone number does accept ext.123 if a business uses an extension. The survey information is sent to the respective database once the information is trimmed and sanitized.

The three Views are accessible from anywhere on the website by the button with the corresponding name in the SiteBanner.vue, which is always visible to the user.

3.7.2 Website Layout

Vue.js is built with single page application's (SPA) in mind. The user accesses the websites tools and capabilities through App.vue, which brings child .vue components upon request. App.vue ensures the top banner and footer components will always render, ensuring cohesive website style and navigation, regardless of the individual page the user is examining.

Vue.js combines HTML, CSS, and Java or TypeScript into a single .vue file format. Each .vue file is a comprehensive component that will render according

to prescribed behavior when called into scope.

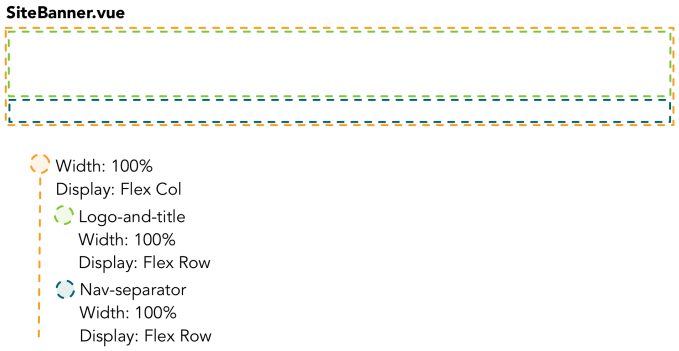

Figure 10: Site Banner

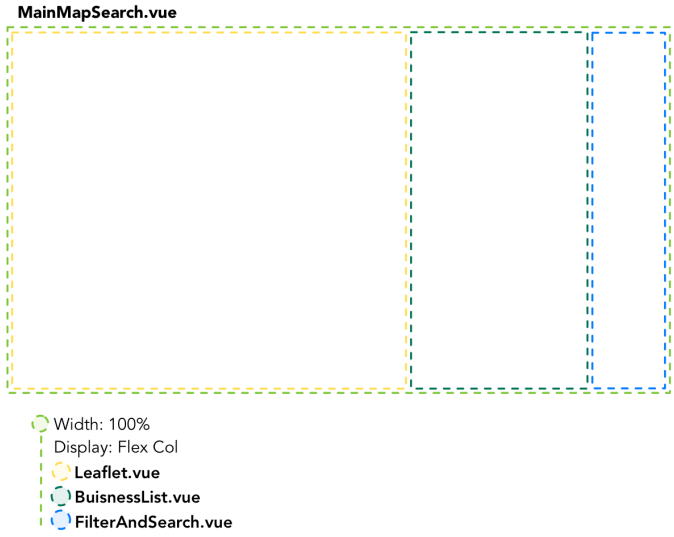

Figure 11: Main Map Search

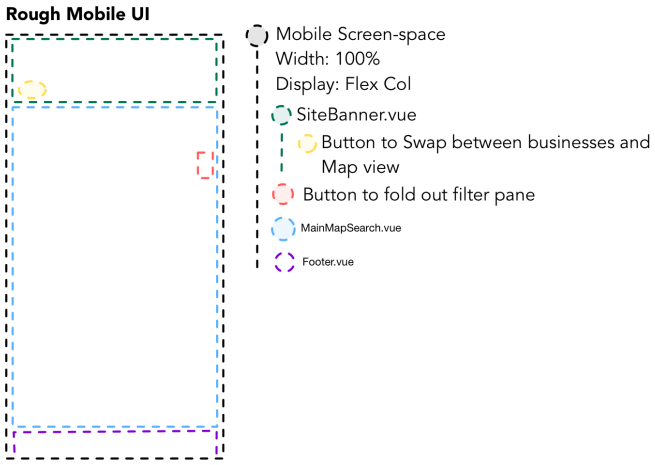

Figure 12: Mobile UI Structure

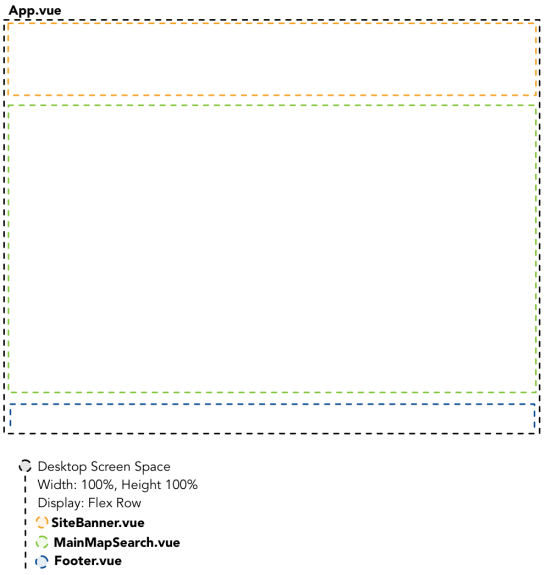

Figure 13: App Interface. SiteBanner.vue, and Footer.vue are always rendered, MainMapSearch.vue renders when it is requested via a router-view. When a user requests a survey page, that component will take MainMapSearch.vue's place.

Depicted above are layout diagrams, showing the size and behavior of the four major components. Vue.js is a flexible web-engine, meaning any of these

components can be reordered or moved without affecting the website as a whole. The width and layout behavior are provided in order to simplify adding to, or moving any of these components during future development.

3.8 Testing

- Surveys & Posting to the databases: Tests were done to ensure that information inserted could not exceed a certain given length and that some information was required to be in a specific format. This was done by trying to submit information that should not be accepted and fixing the issue if it did get submitted.
- Filtering: Different use cases and user testing were used to ensure that the SearchAndFilter.vue and Leaflet.vue were working correctly. This revealed an issue with how BusinessList.vue received business information as changes in Leaflet.vue and SearchAndFilter.vue were not reflected correctly. This led to the changes in how the business information was being passed between each, where it goes from SearchAndFilter.vue, Leaflet.vue, and finally to BusinessList.vue.

4 Human Interfaces

4.1 Color Palette

Figure 14: Rough sketch of the UI depicting the Map Search page, created after final scope and goals was agreed to by both parites.

Many of the colors that appear throughout the website are either pulled from, or based on the colors present in the ENMRD logo. When designing

the user interface, the goal was to both create a modern and professional website, while keeping the colors warm and inviting. To this end, dark blue and green are complimented by the orange any yellow present in the ENMRD logo, while a cream background ties the websites content to the default pallette of OpenStreetMap.

Figure 15: Color Pallet of website using Hex-code for HTML rendering

4.2 Desktop UI

In order to make the website easy to use, the function allowing private landowners to find and contact industry leaders is fully described on the home page. This showcases the map of the surrounding area, the list of businesses that adhere to the given filters, and the filter pane itself. Moreover, the important information is displayed on several different spots, with text bubbles appearing on the map to show business info, as well as the same information appearing on the business info pane. Showing information in more than one spot reinforces the usability of the website for a wide variety of users that may not be as technologically inclined.

The same tenants are present in website navigation. There are more than one way to redirect the user to the home page to search for businesses on the map, and, the navigation buttons directing to both the forestry ENMRD website, and the survey pages are present on any page of the website.

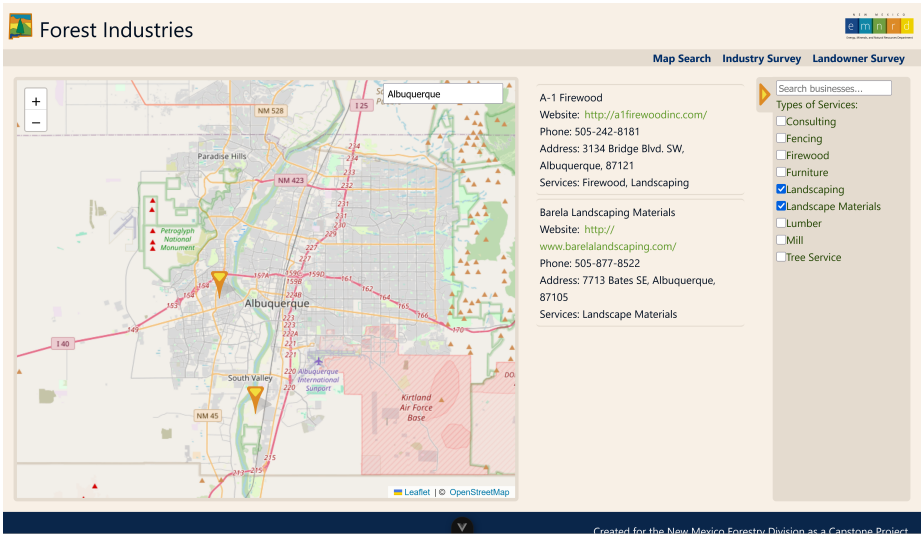

Figure 16: Shows Albuquerque businesses that do Landscaping, has Landscaping materials, or both.

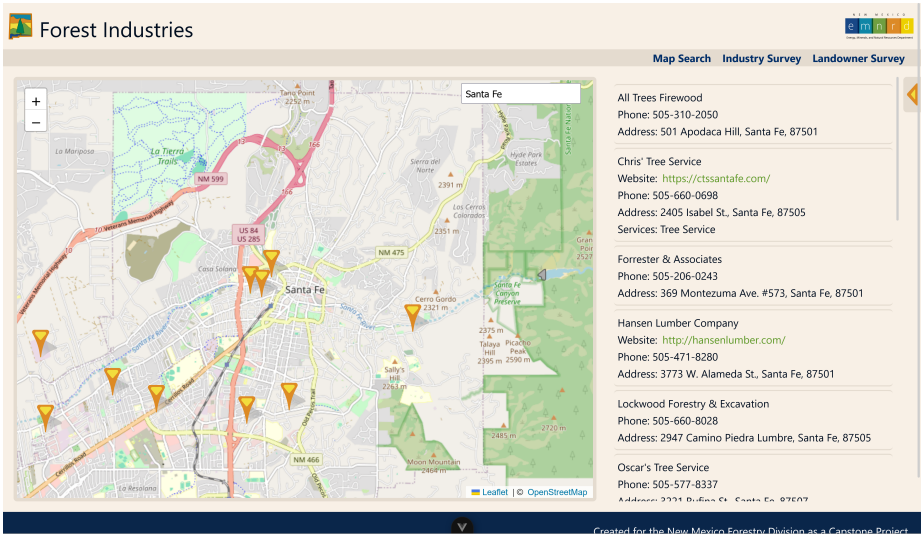

Figure 17: Shows Santa Fe businesses

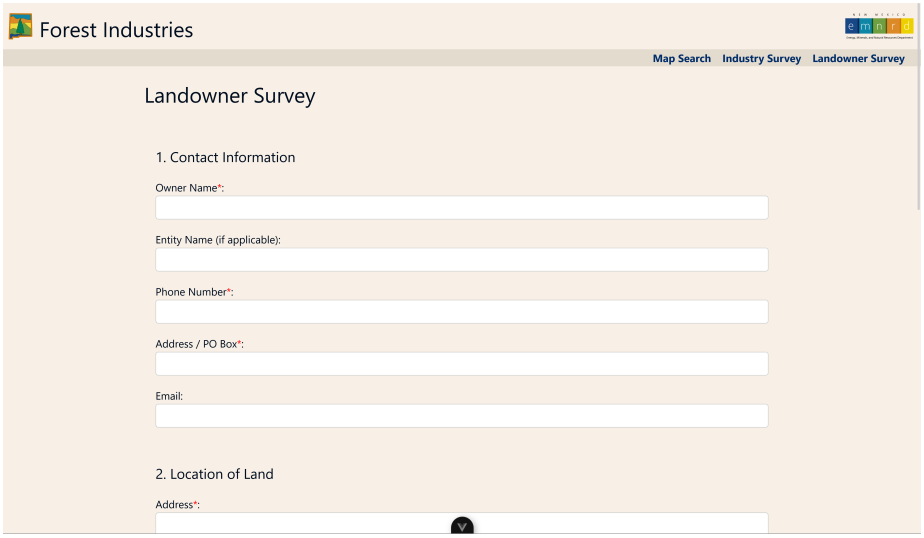

Figure 18: Shows the top half of the Landowner Survey in desktop format.

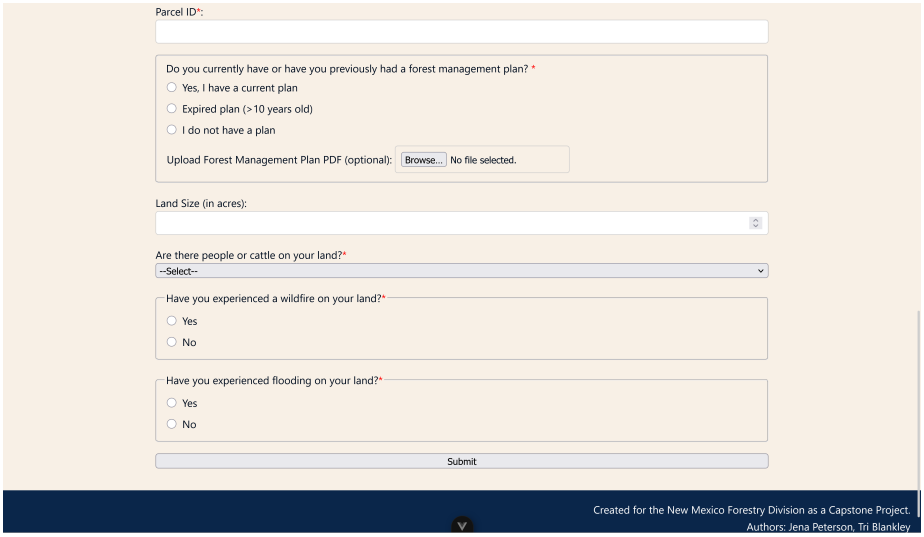

Figure 19: Shows the top bottom of the Landowner Survey in desktop format.

4.3 Mobile UI

The mobile UI this website largely follows the desktop in structure and website navigation. However, to reduce clutter on the screen, and make each section more readable, MainMapSearch.vue was refactored to render only the

map, or the list of businesses, with the filter pane being compressed and folded in by default. A button to the Nav bar was added to swap between this view.

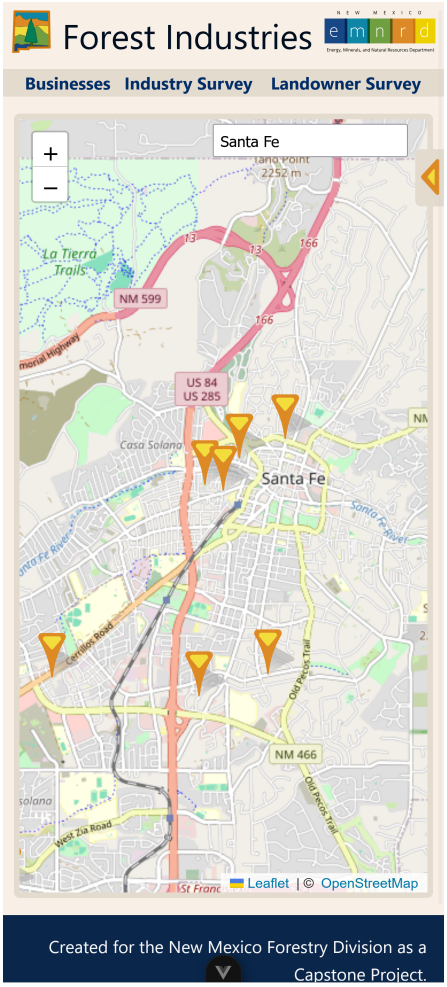

Figure 20: Shows the businesses in Santa Fe

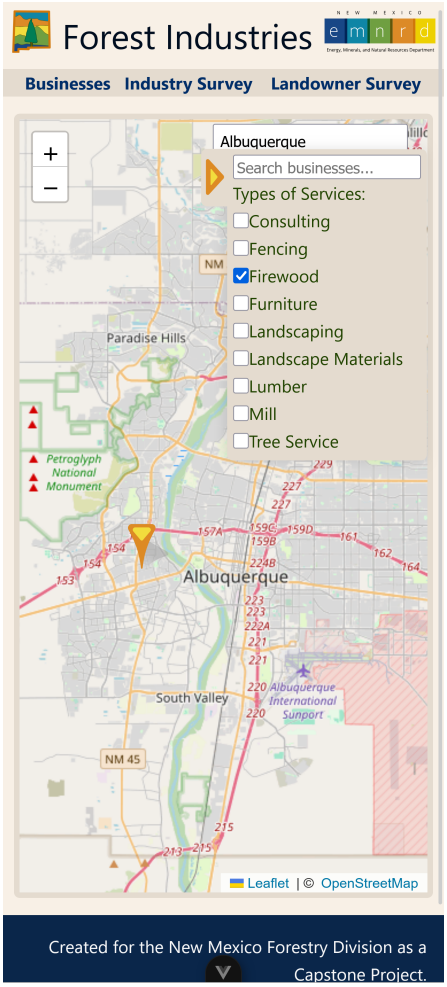

Figure 21: Shows businesses in Albuquerque that sell firewood with filter pane open.

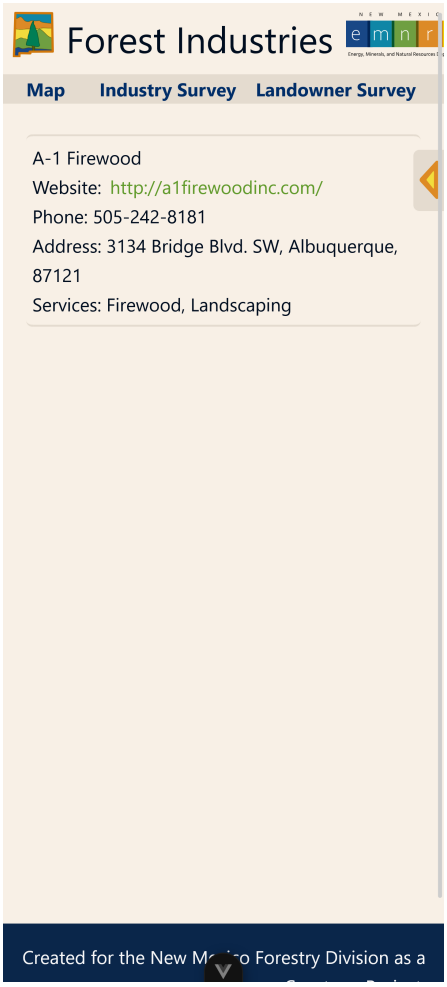

Figure 22: Shows the list of businesses in Albuquerque that sell firewood

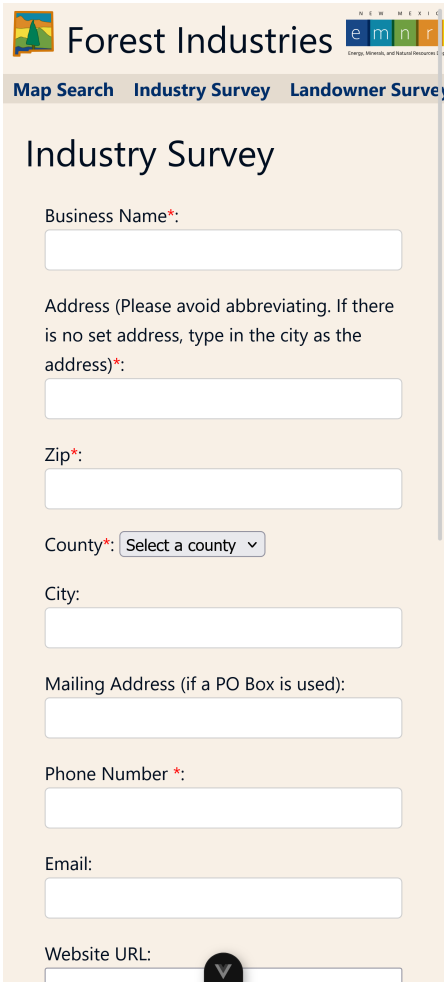

Figure 23: Industry Survey

4.4 UI Testing

- Testing component layout with Vue.js developer tool suite
- Manual usertesting with various scenarios and screen sizes
- External users evaluate user-friendliness
- Changes made based on feedback (home button location, logo placement)
- Cross-browser testing (Chrome, Edge, Firefox differences noted)

  - Scrolls bars show up differently across different browsers. Changes were implemented to stop the scroll bar from taking up too much screen space on Chromium based browsers.

Due to time constraints, the number of UI/UX testers available to look at the project was rather small. To compensate for this, the development team preformed testing using external tools like the Vue.js developer package, as well as several Firefox extensions providing more flexible testing environments. Testing was carried out with the aim on undermining the website based off of architectural decisions such as the behavior of scrollable sections and div render order. During testing, several issues with the map rendering system were found and addressed, as well as several UX issues, particularly when the website is rendered on mobile sized screens.

5 Appendices

5.1 Responsibilities

- Tri Blankley
  - Mobile UI
  - Graphical assets and buttons
  - Website style and CSS
- Jena Peterson
  - Industry & Landowner Databases
  - Database integration with OpenStreetMap
  - Connect survey pages with databases
  - Search, filtering, & business list
- Dominic DOnofrio
  - Docker
  - Setup Front-End of Survey Pages
  - Pen Testing Recommendations in moving forward document.

5.2 Problems Encountered & Remedies

- Desktop to Mobile UI
  - The smaller screen space when on mobile required us to restructure MainMapSearch.vue to only render one component at a time.

  - When this restructure happened, new navitation buttons had to be created. To keep with the website style, it was decided to place these buttons on the top banner, which required slight refactoring to allow SiteBanner.vue to pass values to MainMapSearch.vue in order to control the div state.

- Different browsers rendering space bars differently. This just required additional CSS to ensure scroll bars did not take up too much room.

- Filtering businesses while handling filter requirements from Leaflet.vue and FilterAndSearch.vue

  - The filters need to be applied to several .vue components before the map and business list is refreshed to show the desired changes, special care was taken to ensure every component received the updated filter status
  - While on Mobile UI, the map would not render filtered pins until a page reload occurred. To fix this,a mounted value was added to the Leaflet.vue component to detect a change in filter status

- The Leaflet map will fail to render if static height is not set. To fix this, only give height values render

- Some components need to pass values to one another to make the site reactive in a logical way. This can either be done via a storage package that collects and passes values, or, the website can be structured with parent-child relationships between components. For this project, a parent child structure was chosen, as it reduced the website complexity while providing the necessary functionality.

5.3 Project Status

Requirements and Documentation

The project has successfully met all outlined requirements, delivering a fully functional system that aligns with the objectives of creating a useful tool to connect landowners to industry leaders, while offering a method for the NM Forestry service to collect vital data on landowners. Every component of the codebase is thoroughly documented and commented, ensuring clarity for future developers and maintainers. This documentation extends to both backend logic and frontend design, facilitating seamless updates and troubleshooting.

Database Structure and Deployment

The databases are set up, with separate schemas for industry and landowner data to ensure clean organization and efficient querying. SQL tables will be provided, allowing for straightforward migration and scalability. The system

supports user-generated content, enabling both landowners and industry leaders to submit new entries directly into their respective databases. The entire application is containerized using Docker, simplifying deployment and ensuring consistency across development environments.

User Experience and Visual Design

The UI has been crafted for both functionality and aesthetic appeal, featuring fully vectorized assets and responsive design that adapts to various screen sizes, including mobile devices. Interactive mapping tools, powered by open-source data, dynamically render pins based on industry entries and support filtering for enhanced usability.

Project Milestones

- Completed all outlined project requirements.
- Fully documented and commented codebase.
- Established separate databases for industry and landowner data.
- Implemented user submission functionality for database entries.
- Filtering and search functionality for business data is functional.
- Designed a visually appealing and intuitive UI.
- Ensured responsive design for mobile and desktop compatibility.
- Containerized the application using Docker for easy deployment.
- Integrated open-source mapping tools with dynamic pin rendering.