CEREO Living Atlas

Prototype Project Report

Center for Environmental Research, Education, and Outreach (CEREO)

Living Atlas Development Team

Garoutte, Zachary

Simmons, Jonathan

Gao, Yaru

April 17th, 2025

TABLE OF CONTENTS

I. Introduction

The CEREO Living Atlas is a web-based platform built through collaboration with the Center for Environmental Research, Education, and Outreach (CEREO) at Washington State University. It's intended as a flexible tool for a wide community—researchers, tribal groups, educators, students, and government agencies alike—making it easier to gather, view, and share environmental data relevant to the Pacific Northwest. Providing users with a single, map-focused interface to upload and explore geospatial information. From tracking water quality to monitoring restoration work and community engagement, the platform is geared toward surfacing patterns, highlighting case studies, and guiding decisions with clarity.

For years, environmental datasets were fragmented—tucked away in separate systems, hard to access, and rarely interoperable between institutions. That's a big part of why the Living Atlas came to be. It offers a public-facing, visual interface where users can submit "cards" packed with metadata, photos, location data, and even links to research. These cards are searchable and filterable by category, tag, or custom fields, giving users a more intuitive and visual way to navigate complex datasets.

The platform didn't start out this way. What began as a simple data viewer gradually grew into a fully interactive application with strong backend support. During its alpha phase, a number of useful features were introduced: things like account logins, role-specific permissions, bookmarks, sortable cards, thumbnail previews, and an easier submission workflow for spatial data.

Throughout development, CEREO stakeholders—including faculty, student researchers, and external collaborators—voiced a clear need: the platform had to be user-friendly for those without technical backgrounds, but still robust enough for deeper academic or policy-related work. Their input pushed the team to focus on accessibility and scalability while respecting data ownership. These concerns, in many ways, shaped how the system functions today.

This version of the document reflects the alpha prototype's current status. It summarizes the key requirements, the design and development approach, features completed so far, and the testing methods used to check performance. Also included is a rundown of the project demo shown to mentors, the main feedback that came out of those sessions, and a plan for what's next as development continues.

II. Team Members - Bios and Project Roles

Zachary Garoutte is a software engineering student. He is experienced in working with C, C++, C#, Python, Java, SQL, HTML, and CSS. For this project, his responsibilities have been to create a new database and cloud data service for the project, and to add the ability for users to upload thumbnails to cards in the Atlas.

Yaru Gao is a computer science student. He is experienced in working with C, C++, Python, Java, HTML, CSS, and JavaScript. For this project, his responsibilities have been to refine the frontend UI, fix the bugs of the password reset feature and card display feature, and implement a bookmarking feature that users can use to add, remove or sort their favorite cards.

Jonathan Simmons is a computer science student. He is experienced in C, C++, C#, Python, and Java. For this project, his responsibilities have been to add a sorting feature that can organize the cards in the Atlas by different criteria, such as closest to current location and most recently added, and to add the ability for users to attach files to cards in the Atlas.

III. Project Requirements

This section provides an overview of the features and specifications that must be developed over the course of the development cycle. These requirements were established based on the needs of the client. These requirements may include functional requirements, which are features that are to be implemented, and non-functional requirements, which requires that the project must retain performance under different conditions. There also must be consideration for how the project may evolve over time and how this affects the project requirements.

III.1. Use Cases

The use cases for the application are based on a role-based access control system. Any user has access to the map data and can use search functionality on this data. A registered user is a user who has created an account on the website, and these users can login to their account and bookmark data. An authorized user is a user with authority to add data to the map, and view, edit, or remove this data later. An administrator has authority to grant authorization to add data and can edit or remove any current data on the map. Administrative function is currently planned to be done through the backend, but administrator login may be implemented in the future.

A use-case UML diagram for the Living Atlas is provided in Appendix A.

Add Geospatial Data To Map

Actors	Authorized user
Preconditions	- Be logged in to an authorized user account - On main map page
Postconditions	- Geospatial data and attached information card is added to database and is publicly displayed on map
Path	1. Authorized user clicks Add Data option from homepage 2. User enters geospatial information such as geospatial data types or map coordinates to establish location for data to be stored on

	map 3. User enters relevant information such as study results to attach to these coordinates as a card 4. Optionally upload photo to display on card 5. User submits data addition 6. Data is displayed on map at given coordinates and new card is added
Related Requirements	III.2.1 User Management: Role-Based Access Control III.2.2 Data Management: Geospatial Data Uploading & Editing III.2.3 Map & Visualization Features: Interactive Mapping Interface III.2.3 Map & Visualization Features: Picture Data Upload & Display

View Added Geospatial Data

Actors	Authorized user
Preconditions	- Be logged in to an authorized user account - On main map page
Postconditions	- Information cards corresponding to the geospatial data that was previously added by the logged in user are displayed
Path	1. Authorized user clicks View Added Data option from homepage 2. Information cards added by the user are displayed
Related Requirements	III.2.1 User Management: Role-Based Access Control III.2.3 Map & Visualization Features: Interactive Mapping Interface

Edit Added Geospatial Data

Actors	Authorized user
Preconditions	- Be logged in to an authorized user

	account - On main map page
Postconditions	- Edit the text and/or photo on an information card previously added by the logged in user
Path	1. Authorized user clicks View Added Data option from homepage 2. Information cards added by the user are displayed 3. Click Edit option on card to be edited 4. User enters updated information for card 5. Optionally user uploads new photo for card 6. User submits edit 7. Information and photo is publically updated in real time on the card
Related Requirements	III.2.1 User Management: Role-Based Access Control III.2.2 Data Management: Geospatial Data Uploading & Editing III.2.3 Map & Visualization Features: Interactive Mapping Interface III.2.3 Map & Visualization Features: Picture Data Upload & Display

Remove Geospatial Data From Map

Actors	Authorized user
Preconditions	- Be logged in to an authorized user account - On main map page

Postconditions	- Geospatial data and attached information card previously added by the logged in user is removed from the map
Path	1. Authorized user clicks View Added Data option from homepage 2. Information cards added by the user are displayed 3. Click Delete option on card to be deleted 4. User confirms deletion 5. Geospatial data on map is removed from map and the attached card is removed
Related Requirements	III.2.1 User Management: Role-Based Access Control III.2.3 Map & Visualization Features: Interactive Mapping Interface

Bookmark Data

Actors	Registered user of any authorization
Preconditions	- Be logged in to an account - On main map page
Postconditions	- Information card is saved in user's bookmarks
Path	1. User clicks bookmark on card to be bookmarked 2. Card is saved in user favorites
Related Requirements	III.2.2 Data Management: Data Filtering & Search

Un-Bookmark Data

Actors	Registered user of any authorization
Preconditions	- Be logged in to an account - On main map page
Postconditions	- Information card is saved in user's bookmarks

Path	1. User clicks bookmark on a card that is bookmarked 2. Card is removed from user favorites
Related Requirements	III.2.2 Data Management: Data Filtering & Search

Access All Bookmarked Data

Actors	Registered user of any authorization
Preconditions	- Be logged in to an account - On main map page
Postconditions	- Information cards previously bookmarked by the logged in user are displayed
Path	1. User clicks View Bookmarks option from homepage 2. Information cards bookmarked by the user are displayed
Related Requirements	III.2.2 Data Management: Data Filtering & Search

Search (updated)

Actors	Any user
Preconditions	- On main map page
Postconditions	- Display information cards based on date, location, and other sort and filter options
Path	1. Optionally click sort option to sort list of cards 2. Optionally click filter option to filter list of cards 3. Display cards best matching the search criteria
Alternate Path	1. User clicks search bar 2. Type keywords to be matched with cards in database 3. Display cards relating to keywords

	4. Optionally click sort option to sort list of cards 5. Optionally click filter option to filter list of cards 6. Display cards best matching the search criteria
Related Requirements	III.2.2 Data Management: Data Filtering & Search

Reset Password

Actors	Registered user of any authorization
Preconditions	- On login page or profile page
Postconditions	- The user's password is updated
Path	1. User clicks Forgot Password option 2. A one-time verification link is sent to the email attached to the user's account 3. Click link in email account 4. Enter new password 5. User submits password 6. Login password is updated to new password
Related Requirements	III.2.1 User Management: User Registration & Authentication

III.2 Functional Requirements

The CEREO Living Atlas is a web-based application developed to assist researchers, tribal communities, and government agencies in visualizing and sharing critical environmental data. To ensure the system remains accessible, efficient, and scalable, a structured set of functional requirements has been established.

These functional requirements define the system's essential capabilities, outlining what the application must achieve to effectively meet user needs. They are categorized into three primary modules: **user management, data management, and mapping visualization.**

Each requirement is aligned with stakeholder needs and assigned a priority level based on its significance:

- **Priority Level 0:** Essential, non-negotiable functionality.
- **Priority Level 1:** Important but not critical features.

- **Priority Level 2:** Optional enhancements or future upgrades.

III.2.1 User Management

- **User Registration & Authentication:**
  - The system must allow users to create accounts and log in securely.
  - Users should be able to reset passwords via email authentication.
  - Emails must be reliably forwarded from the application's Gmail account to WSU emails.
  - Source: CEREO's need for controlled data access and email functionality improvements.
  - **Priority:** Level 0 (Essential)
- **Role-Based Access Control:**
  - The system must differentiate user roles (e.g., researchers, administrators, public users).
  - Certain data upload and modification features must be restricted to authorized users.
  - There is currently no admin login, but this may be required in the future.
  - Source: Stakeholder request for future expansion options.
  - **Priority:** Level 1 (Desirable)

III.2.2 Data Management

- **Geospatial Data Upload & Editing:**
  - Users must be able to upload geospatial files (e.g., shapefiles, GeoJSON) instead of hardcoding data.
  - The system should support additional data types, such as watershed boundaries.
  - Polygons representing geospatial data should be customizable, such as color coding to represent different environmental factors.
  - Source: Client request for flexible spatial data storage and visualization.
  - **Priority:** Level 0 (Essential)
- **Data Filtering & Search:**
  - Users should be able to filter datasets based on date, location, and environmental metrics.
  - A search function must allow users to locate specific data points easily.
  - Users should be able to bookmark data to make the data easily accessible at a later time.
  - Source: Stakeholder need for enhanced data navigation.
  - **Priority:** Level 1 (Desirable)

III.2.3 Map and Visualization Features

- **Interactive Mapping Interface:**
  - The map must dynamically display geospatial data and be updated in real-time.

    - Users should be able to toggle different map layers (e.g., satellite, terrain).
    - The map must include a legend of symbol meanings, categories, and active layers.
    - Source: Usability improvement request from researchers and policy analysts.
    - **Priority:** Level 0 (Essential)
- **Picture Data Upload & Display:**
    - Users must be able to upload images associated with geospatial data points.
    - Ensure smoother handling of image uploads to avoid failures.
    - Source: Client feedback on frontend usability.
    - **Priority:** Level 0 (Essential)
- **Performance Optimization:**
    - The website should be able to work faster and handle a larger user base.
    - Reduce delays in loading data points on the map.
    - Source: Client's primary goal for enhancement.
    - **Priority:** Level 0 (Essential)

III.3. Non-Functional Requirements

Non-functional requirements are aspects such as performance, security, reliability, and scalability. While the functional requirements define what the system must do, non-functional requirements determine how efficiently and effectively it operates under different working conditions.

To support its expanding user base and large datasets, the CEREO Living Atlas must ensure data integrity, high availability, and an intuitive user experience. Meeting long-term stakeholder expectations requires a strong focus on efficiency, security compliance, and system extensibility.

These requirements are categorized into key areas, including performance, security, cross-platform compatibility, reliability, and future scalability. Each is assessed and prioritized based on its influence on system usability and long-term sustainability.

III.3.1. Performance & Scalability:

- The system shall support 500+ concurrent users without performance degradation.
- The backend should be optimized to handle large datasets efficiently.
- Source: Client's requirement for scalability.
- Priority: Level 0 (Essential)

III.3.2. Security & Authentication:

- Email forwarding issues must be resolved to allow WSU emails to receive system messages.
- Future improvements should consider adding email-based user verification.

- Source: Client's security concerns.
- Priority: Level 1 (Desirable)

III.3.3. Cross-Platform Compatibility:

- The web application shall function on modern browsers (Chrome, Firefox, Edge).
- The system should support mobile access with a responsive design.
- Source: User accessibility requirements.
- Priority: Level 1 (Desirable)

III.3.4. Reliability & Uptime:

- The system shall maintain a 99.5% uptime with failover mechanisms.
- Regular database backups must be implemented to prevent data loss.
- Source: Client expectation for high-availability services.
- Priority: Level 0 (Essential)

III.3.5 Extensibility & Future Integrations:

- The system should be modular to allow future integrations (e.g., external GIS platforms).
- API endpoints should be designed for third-party compatibility.
- Source: Long-term system evolution considerations.
- Priority: Level 2 (Stretch Goal)

III.3.6 Improved Application Accessibility:

- Develop a standalone WinForms application that wraps the CEREO web application.
- Users can launch the app via a desktop shortcut instead of running npm start in a terminal.
- The application can be easily installed by downloading and extracting a ZIP file from GitHub, providing a more intuitive setup for non-technical users.
- Priority: Level 1 (Desirable)

III.4. System Evolution

The development of the CEREO Living Atlas relies on several key assumptions, including the expectation that the current application holder will continue hosting the web application on their server. It is also assumed that the team's understanding of client needs is aligned with project objectives and that the planned improvements are correctly oriented to address the identified challenges. Additionally, the necessary permissions are expected to be granted for testing and development on the existing application.

Several risks could impact the project's success. As part of the development process, we will be enhancing user role management functionalities, including permission controls and access restrictions. Modifying these core features may inadvertently introduce security gaps, disrupt existing workflows, or create unintended access issues. Additionally, testing functionalities, such

as adding a new polygon point to the map, may inadvertently alter or disrupt CEREO's existing data. Furthermore, the introduction of new features or enhancements to existing ones may introduce unforeseen bugs due to compatibility issues or conflicts with current system components, potentially causing temporary disruptions in system functionality.

Certain client requirements remain unclear, including the specific functionality of image attachments and the intended use of new polygon features, such as shape options and flexibility. Understanding how polygons support environmental research is necessary for precise implementation. Additionally, the goal of seamless student and faculty contributions needs clarification—whether it pertains to user-friendliness, system scalability, or enhanced access controls. These aspects will be addressed in future client meetings.

IV. Solution Approach

IV.1. System Overview

The functionality of the application's system should be focused on providing a user-friendly experience for all users that will be navigating through the Living Atlas, while also accommodating support for features for more technical users, such as adding complex geospatial data types to the database. To achieve an intuitive user interface, we will implement features found on other GIS-based websites like Zillow which achieve seamless search and filtering for items on a map while being easy-to-use and compatible across all devices. For users adding data to the database, the system should support upload of shapefiles and/or GeoJSON files to represent the geospatial data as shown on the map. The system will be designed such that these geospatial data files can be transferred from a user upload at the application frontend to the backend and into the database, which the frontend will read from and display its data on the map.

IV.2. Architecture Design

IV.2.1. Overview

The Living Atlas is designed with a three-layer architecture, comprising the frontend, backend, and database. The frontend, developed using React.js, delivers an interactive and user-friendly interface, enabling seamless map visualization and intuitive user interactions. On the backend, FastAPI acts as the core intermediary, efficiently managing incoming API requests, authentication, and data processing. Supporting this all is a PostgreSQL database, enhanced with PostGIS, which securely and effectively stores geospatial data, user credentials, and metadata related to uploaded datasets. This modular approach ensures that each component can be updated or modified independently, enhancing the system's scalability, flexibility, and long-term maintainability.

A diagram of the basic subsystem layers of the architecture can be found in Appendix B.

IV.2.2 Subsystem Decomposition

The system is structured into three core subsystems: the frontend, backend, and database. The frontend handles data visualization, processes user interactions, and provides tools for data submission. To enhance mapping capabilities, it leverages libraries such as Mapbox and Leaflet.js. The backend manages user authentication, enforces role-based access, validates data uploads, and delivers geospatial data through API endpoints. Meanwhile, the database efficiently organizes structured data, ensuring rapid retrieval and optimized querying through spatial indexing.

Frontend Subsystem

The frontend subsystem is responsible for managing user interactions and rendering geospatial data. It consists of key components such as the homepage, an interactive map, and data submission forms. By leveraging API calls, it seamlessly communicates with the backend to ensure smooth data processing and storage. A diagram of the frontend subsystem can be found in Appendix C.

Backend Subsystem

Serving as the system's processing core, the backend manages user authentication, enforces access control, and validates geospatial data submissions. Positioned as the bridge between the frontend and the database, it ensures secure and efficient data flow. To optimize performance, various algorithms are implemented to streamline API request handling and minimize response times. A diagram of the backend subsystem can be found in Appendix D.

Database Subsystem

The database subsystem is responsible for securely storing crucial system data, including user credentials, authentication records, and geospatial datasets. Built on PostgreSQL with PostGIS extensions, it is optimized for handling spatial queries with high efficiency. Its scalable architecture enables rapid data retrieval, advanced filtering, and indexing to ensure seamless access to stored information. A diagram of the database subsystem can be found in Appendix E.

IV.3. Data Design

The database design consists of structured tables to store geospatial data, user information, and role-based permissions. The Geospatial Data Table contains dataset entries, including spatial attributes, metadata, and associated files (e.g., shapefiles, GeoJSON). The User Table maintains account credentials, authentication tokens, and access levels. The Permissions Table defines role-based access to ensure only authorized users can upload or modify data. Data indexing and relational integrity are enforced through PostgreSQL with PostGIS, allowing for optimized querying and retrieval of large-scale environmental datasets.

There will be two major data structures present in the system database: The data structure containing all geospatial data points (including attached card information), and the data structure containing the account information of all users. Each object in the geospatial data structure will contain a shapefile so that the data can be displayed properly on the map in the

correct location, and the card information for that data point, which can include the name of the user who added the data, the user's email, the organization who collected study data, a link to the study dataset, the title of the linked dataset, a description of the data set, the data category, and search tags. Each object in the account information data structure will contain a user ID, username, email, encrypted password, and the user's authorization level (view only, authorized to add data, or admin).

IV.4. User Interface Design

The CEREO Living Atlas provides an interactive web interface for users to explore and manage environmental data. The homepage serves as the main point of interaction. Users can search, filter, and navigate data efficiently, while authorized users can contribute and manage geospatial information. The features described here align with those listed in the Use Cases section.

The homepage offers several key features. Users can search for data using filters in the top navigation bar. Geospatial data can be added through the "Add" button in the cards section. Viewing data is straightforward, with coordinates and shapes displayed on the left-side map and corresponding cards on the right. Users can also remove data from the same card section. Bookmarked data is accessible in this area as well. For diagrams illustrating these features, refer to Appendix F.

Editing data is available by clicking "Learn More" on a card, which opens a detailed window containing an edit option. Additionally, this window also allows authorized users to delete a card, update information and attached files as needed. For visual references of this feature, see Appendices G and H.

The user center includes account management functions. Password resets require entering the current password and a new one. This ensures security while keeping the process simple. A diagram of this feature can be found in Appendix I.

The homepage navbar provides access to key pages, including the registration page and administrator dashboard. The registration page allows non-logged-in users to fill out a form requesting an account. Submitted requests are sent to the admin panel, where an administrator can approve or deny them. For a diagram of this process, see Appendix J and K, respectively.

The administrator dashboard displays a list of all registered users in the Living Atlas. Admins can manage user accounts by modifying access levels or deleting accounts. This ensures proper role-based access control within the system. For a detailed view of the admin dashboard, refer to Appendix L.

V. Test Plan

To validate the functionality and robustness of the CEREO Living Atlas alpha prototype, a combination of unit, integration, and manual tests were conducted across both frontend and backend systems. Testing focused on verifying core features such as card creation, thumbnail

uploads, user role access, and data filtering. Continuous integration checks ensured the backend builds successfully on every commit. Manual testing also confirmed proper rendering and responsiveness across modern browsers. Together, these tests provide confidence in the stability of the current prototype while highlighting key areas for further refinement in the next development phase.

V.1. Unit Testing

Our team will follow all standard unit testing procedures. Most unit testing will focus on code developed by previous teams, as new development is mostly built using these previously developed units. The previously developed units to be tested include loading a card from the database, adding or deleting a card from the database, adding or deleting an account, submitting a form, and loading the map from Mapbox. It is crucial that these previously developed units are tested as the database host has been switched to Microsoft Azure and these units must still function with the new database for the overall system to function correctly. Some newly developed units to be tested include changing an account's password, sending a reset password email, and loading files and images from the database.

V.2. Integration Testing

Given the multi-layer design of our application, consisting of a FastAPI backend, React Front end and integration with google cloud storage- the integration testing focuses on validating the flow of data between these layers. Unlike a monolithic structure, this modular setup introduces complexities in testing complete workflows and can cause issues with the flow of data between the layers.

The development team will primarily rely on manual testing within local and staging environments to replicate integrated scenarios like form submissions involving file and thumbnail uploads, database entry creation, and frontend content display. At this stage, automated integration testing remains limited due to the complexities of mocking Google Cloud APIs and maintaining database state across different layers. However, specific endpoints may still be tested using tools such as Postman or pytest with FastAPI's TestClient.

Although the ultimate goal is comprehensive end-to-end integration testing across all components, certain elements—like frontend thumbnail display logic—may be tested in isolation when backend deployment or effective mocking proves challenging.

V.3. System Testing

V.3.1. Functional testing:

Functional testing is carried out manually by developers, following the project's Requirements and Specifications document. Each functional requirement is matched with a specific test case. This includes:

- Creating a card with all metadata fields

- Uploading data files and thumbnail images
- Retrieving and displaying card data, including image previews
- Downloading uploaded files
- Deleting cards and confirming cascade deletions (files and thumbnails)

Tests are validated through the browser interface and network tools like Chrome DevTools or Postman. Any failed tests are documented with clear reproduction steps and assigned back to the original developer for resolution. As the application evolves, the testing plan will be updated accordingly.

V.3.2. Performance testing:

Performance testing targets two main areas: backend response time and frontend rendering speed.

Backend testing includes:

- Measuring FastAPI response times with large payloads (e.g., multi-megabyte uploads)
- Tracking latency when accessing files and thumbnails stored on Google Cloud

Frontend testing includes:

- Monitoring load times when rendering pages with numerous cards and images
- Evaluating performance on lower-end devices or slower networks

Manual observations are supported with browser profiling tools like Lighthouse and Chrome DevTools. While not benchmarked against strict numerical thresholds, performance is assessed qualitatively based on overall responsiveness and user experience.

V.3.3. User Acceptance Testing:

User acceptance testing will be conducted exclusively by the client. Instead of structured forms, feedback will be gathered through direct meetings.

A. Process

- The client will test key workflows, including creating, viewing, editing, and deleting cards.
- Particular focus will be placed on new features, such as image upload and thumbnail display.
- Observations and feedback will be shared in scheduled meetings.

B. Revision Process

- Developers will document and categorize feedback as bugs, feature requests, or general improvements.
- Actionable items will be tracked in the issue management system (e.g., GitLab Issues).

- Any necessary changes will be implemented and deployed to staging before final release.

The final iteration will be considered complete once the client confirms that all key features function as expected and the user experience meets their needs.

VI. Alpha Prototype Description

The alpha prototype of the CEREO Living Atlas represents a major step forward in functionality, usability, and technical integration. The project team has focused on core features critical to user experience, including card creation with image support, interactive data visualization, and data filtering mechanisms. These systems were chosen based on both stakeholder feedback and project prioritization goals.

The architecture includes a FastAPI backend, PostgreSQL database, React-based frontend, and Google Cloud Storage (GCS) integration for handling media uploads. The alpha prototype includes successful implementation of most high-priority subsystems and the successful integration of these components into a cohesive application.

VI.1. Backend Subsystem – FastAPI & PostgreSQL

VI.1.1. Functions and Interfaces Implemented

- **User Role Management**: The backend enforces logic for distinguishing between public users, registered users, and authorized users, with appropriate access permissions.
- **Card Creation Endpoint** (`/uploadForm`): Accepts multipart form data, including title, description, tags, file uploads, and geographic coordinates.
- **Thumbnail Upload**: Supports optional image upload. If an image is not provided, a default GCS thumbnail is applied. Public URLs are saved in the database.
- **Card Retrieval** (`/allCards`): Returns all cards in the system, joined with associated tags, file metadata, and user information.
- **Filtering & Sorting**: Endpoints like `/filterByTag`, `/sortByDate`, and keyword-based search allow refined access to environmental data.
- **Data Deletion**: Authorized users can delete their own uploaded cards, with proper cascade handling for thumbnails and file links.

Remaining Work:

- Admin-specific backend features (e.g., audit logs, user role editor).

- Automated email verification and token expiration handling.

VI.1.2. Preliminary Tests

- Backend was tested using **Swagger UI**, **Postman**, and manual `curl` requests.
- Thumbnail upload functionality was tested with and without image input to confirm fallback to default.
- Queries to `/allCards` returned correct JOINed data including tags and file paths.
- CI/CD pipeline on Render passed builds consistently, with deployments monitored through webhook activity logs.

VI.2. Frontend Subsystem – React Web Application

VI.2.1. Functions and Interfaces Implemented

- **Card Creation Modal (`FormModal.js`)**: Users can input all metadata and upload optional thumbnail images. Input validation and user feedback are provided.
- **Card Display (`Card.js`)**: Dynamically displays title, description, organization, file links, thumbnail, tags, and date.
- **Search, Filter, and Sort Features**: Implemented via dropdowns, input fields, and buttons to allow browsing through many cards.
- **Bookmarking System**: Allows registered users to save and later view selected cards.
- **Responsive UI**: Fully mobile-accessible layout with modals and sidebar-style filtering.

Remaining Work:

- Image preview in the modal before form submission.
- Possible full overhaul of frontend design

VI.2.2. Preliminary Tests

- Modal input validation was confirmed through simulated form submissions with missing or invalid values.
- All cards retrieved were rendered successfully, including those missing thumbnails (default image was shown).

- Filter/sort controls dynamically updated displayed cards as expected.
- Manual testing confirmed responsive behavior across Chrome, Firefox, and Edge browsers.
- Frontend deployed to **Netlify**, build tested successfully using Git-based previews.

VI.3. Cloud Storage & File Handling

VI.3.1. Functions and Interfaces Implemented

- Integration with Google Cloud Storage for uploading user-submitted files and thumbnail images.
- Image URLs are fetched from GCS and made accessible via public buckets.
- A default thumbnail is uploaded once and reused for cards without an image.

Remaining Work:

- Add expiration-based cleanup logic for unused files.
- Advanced error handling for storage permissions or failed uploads.

VI.3.2. Preliminary Tests

- Uploaded thumbnails were verified via the GCS dashboard.
- Links stored in the database were directly tested in the browser for availability.
- A "broken upload" simulation was run to ensure fallback to the default thumbnail.

VII. Alpha Prototype Demonstration

To begin our demonstration, we showed our clients the various UI enhancements we had made to the Living Atlas, including collapsable panels for the cards and filtering features, updated modals for the card upload and profile features, and a larger, centered map. We walked our clients through the process of uploading a card, including how to upload a thumbnail to a card, and showed how this new card would appear in the Atlas. Resetting an account password was attempted during the demonstration, but an unknown bug prevented the password reset from submitting. This bug will be fixed in future development. The sorting features were also demonstrated, and we showed our clients how nearby cards would appear first when searching by proximity, and the newly added test card from our upload demonstration would appear first when searching by recently added. Bookmarking was demonstrated as well, however, several bugs were found when using this feature. These

include bookmarks disappearing if the map is moved, or the card's ID is not able to be fetched from the database in some circumstances. These bugs will also be fixed in future development. We explained to our client our unfinished work for future sprints, including file attachments, map layer toggling, custom polygons, and bug fixes to currently implemented features.

Our client was impressed by the thumbnail feature and UI enhancements, but would like the bookmarking feature to be fully functioning in the near future. Some other suggestions our client had was to make all hyperlinks attached to cards clickable, to allow admins to add a base layer, and to limit the description in a card preview to only a few lines so that one card would not take up too much space in the list of cards.

Our client had no further questions.

Below is the link that leads to the demonstration of our Prototype to the clients.

Video Link: https://www.youtube.com/watch?v=iFYYRm5QUYY

VIII. Future Work ‘

This semester's work on the Living Atlas involved the app environment, fixing major bugs, and polishing existing feature logic - including switching database deployment, resetting frontend and backend services, fixing card display and reset password issues, and implementing the bookmark feature.

In the next semester, we plan to complete the following pending tasks and enhancements:

- Enhance map to display all cards on homepage load
- Show a star icon at the map location of bookmarked cards
- Filter map to show only favorites when toggled
- Improve frontend UI and add more interactive frontend features to enhance user experience
- Implement togglable map layers like watershed boundaries
- Enable user-drawn polygons with customizable properties
- Fix file upload errors and ensure correct attachment linking
- Allow cards to be clicked and lock the map to the location of its corresponding pin

These will be completed in future sprints with regular progress.

IX. Glossary

Authentication & Authorization:
Authentication verifies a user's identity, while authorization controls their access and permissions.

API (Application Programming Interface):
A set of protocols enabling software applications to communicate and integrate with external services.

Backend:

The server-side logic of an application responsible for processing requests, handling data storage, and managing business logic.

Chrome DevTools:
A set of web developer tools integrated in Google Chrome that can be used to inspect and debug web pages and monitor performance

Concurrent Users:
The number of users accessing the system simultaneously without performance issues.

Continuous Integration:
A software development practice where code changes are automatically tested and integrated into the repository as often as possible

Database:
A structured collection of data stored electronically, which in this case includes geospatial data, user credentials, and metadata.

Data Filtering & Search:
A feature that allows users to refine and retrieve geospatial data based on specific criteria.

Data Integrity:
Ensures stored data remains accurate, consistent, and reliable over time.

FastAPI:
A modern web framework for building APIs with Python, known for its speed and ease of use.

Frontend:
The part of the application that users interact with directly, typically consisting of a graphical user interface (GUI).

GeoJSON:
A format for encoding geographical data structures using JSON

Geospatial Data:
Information that includes geographic location attributes, often represented using coordinates and spatial features.

Geospatial Mapping Interface:
An interactive system that displays and analyzes location-based data on a map.

GIS (Geographic Information System):
A system designed to capture, store, manipulate, analyze, and display spatial or geographic data.

Google Cloud:
A cloud storage service that allows for storage of data accessible over the internet

Leaflet.js:
An open-source JavaScript library used for interactive maps.

Lighthouse:
An open-source Google-created developer tool that can run on a web page and generate a performance report with advice on how to improve performance

Living Atlas:
A geospatial web application designed for visualizing and sharing environmental data, focusing on water quality in the Columbia River Basin.

Mapbox:
A mapping platform providing tools for designing and implementing custom maps.

Metadata:
Data that provides information about other data, such as descriptions, timestamps, and ownership details.

Microsoft Azure:
A cloud computing platform offering database hosting

Netlify:
A platform to host and manage web projects

PostGIS:
A spatial database extender for PostgreSQL that enables advanced geospatial queries.

PostgreSQL:
An open-source relational database system used for managing structured data.

Postman:
An API development tool that can be used to build and test APIs

pytest:
A Python testing framework that supports unit tests and integration tests, which can be used to test APIs

React:
A JavaScript library for building user interfaces

Role-Based Access Control (RBAC):
A security model that restricts system access based on user roles and permissions.

Scalability:
The system's ability to handle growing user activity and data without performance loss.

Shapefile:
A common geospatial vector data format used for geographic information system (GIS) applications.

Spatial Indexing:
A technique used in databases to optimize spatial queries, improving the efficiency of geographic data retrieval.

Swagger UI:
A web-based interface that allows for testing of APIs directly from the browser.

Third-Party Compatibility:
Enables integration with external applications and datasets via APIs or standard formats.

User Authentication:
The process of verifying the identity of users before granting access to the system.

X. References

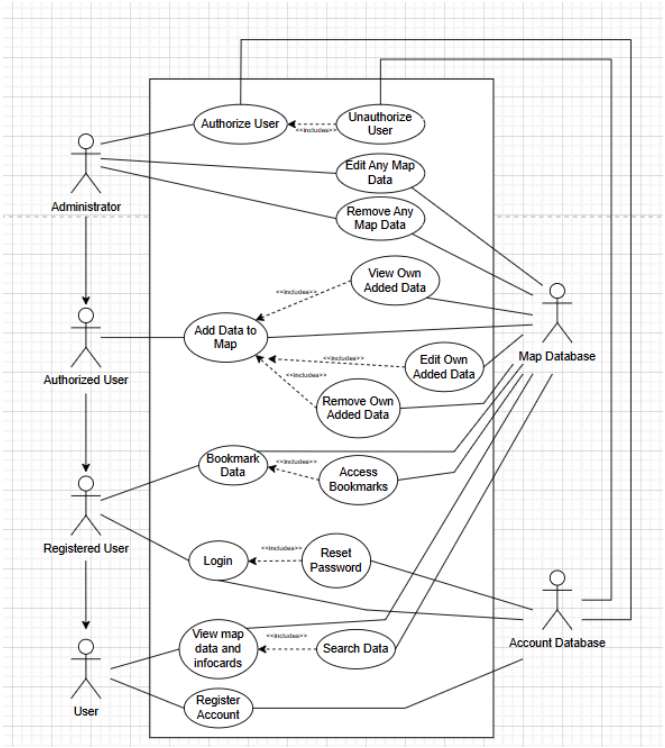
Bruegge, Bernd., Dutoit, Allen H.. Object-oriented Software Engineering: Using UML, Patterns, and Java. United Kingdom: Prentice Hall, 2010.

Lethbridge, Timothy Christian., Laganière, Robert. Object-oriented Software Engineering: Practical Software Development Using UML and Java. United Kingdom: McGraw-Hill Education, 2005.

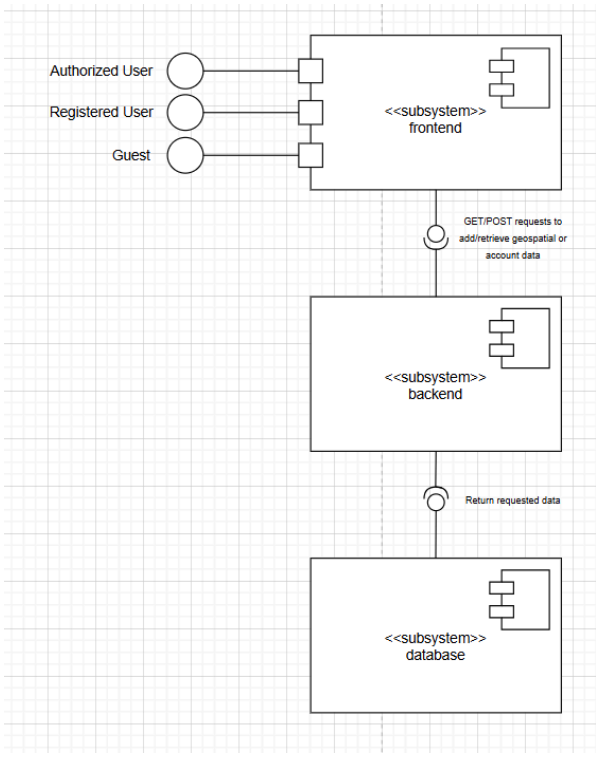
Li, Eldon Y. "Software testing in a system development process: A life cycle perspective." *Journal of Systems Management* 41, no. 8 (1990): 23-31.

XI. Appendices

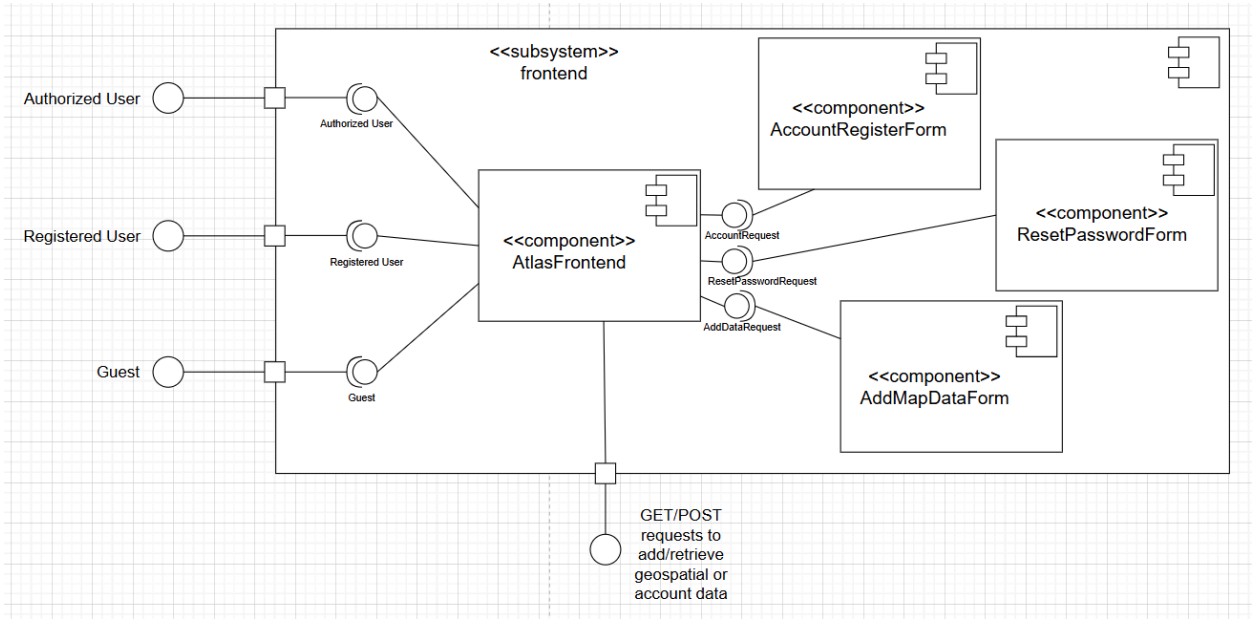
Appendix A: Use Case Diagram



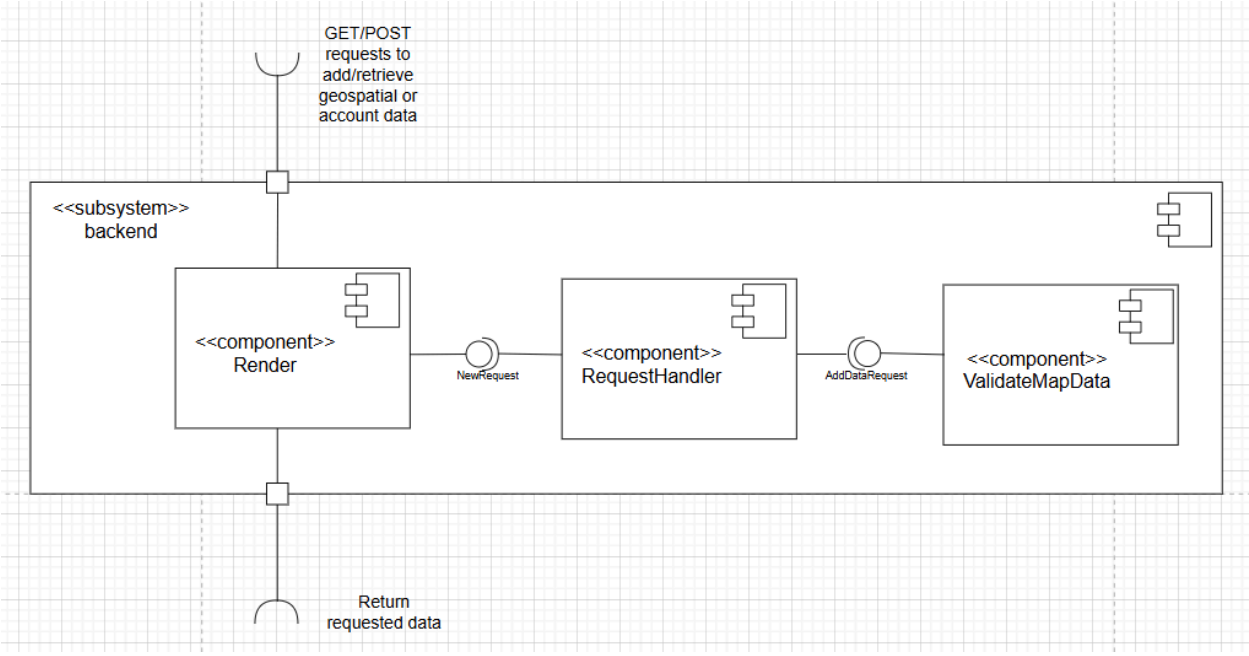
Appendix B: Architecture Design Overview Diagram



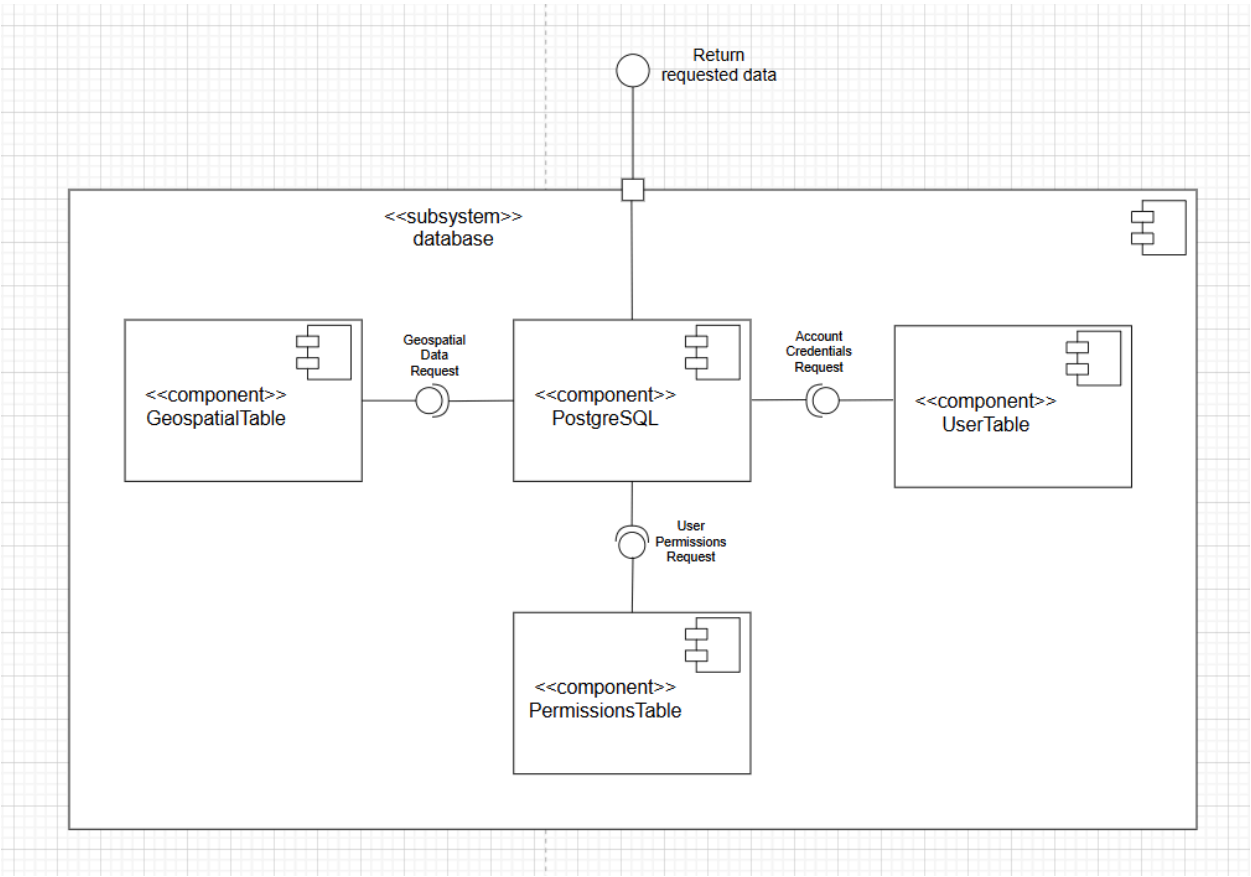
Appendix C: Frontend Subsystem Diagram



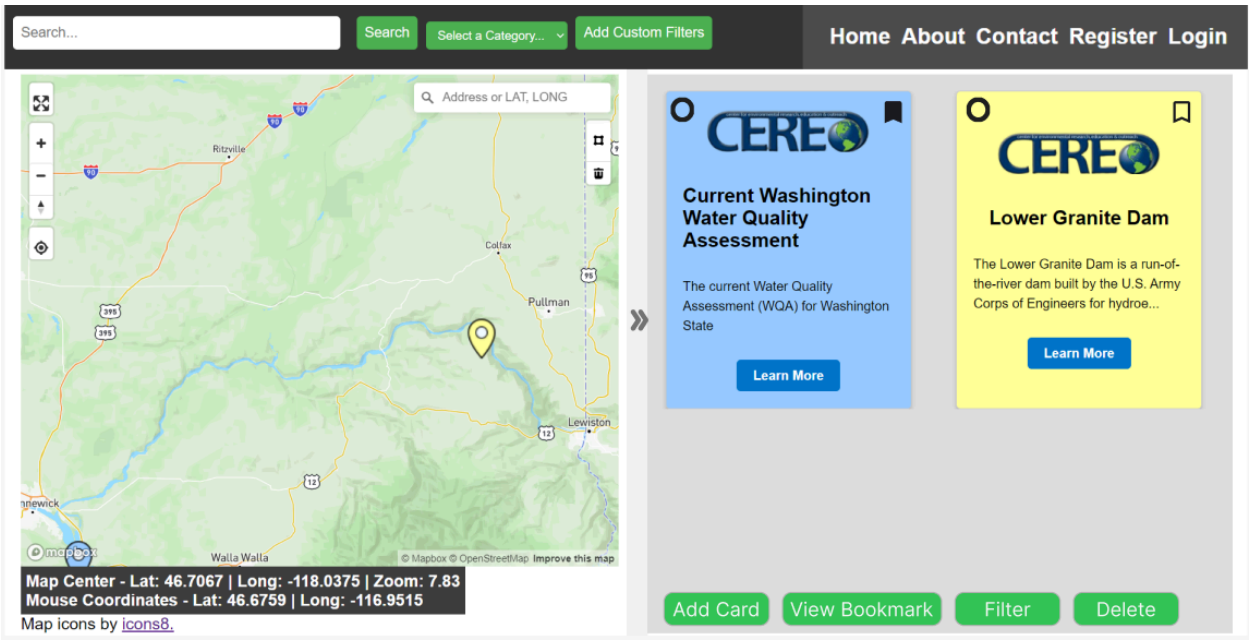
Appendix D: Backend Subsystem Diagram



Appendix E: Database Subsystem Diagram

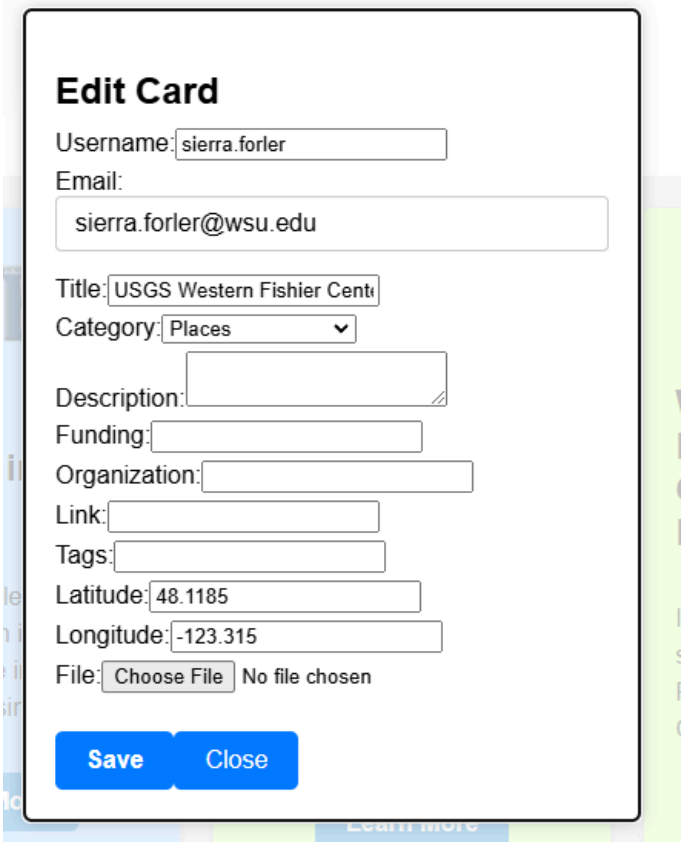


Appendix F: Mock-up diagram of the homepage, illustrating search, filtering, data addition, viewing, removal, and bookmark access.

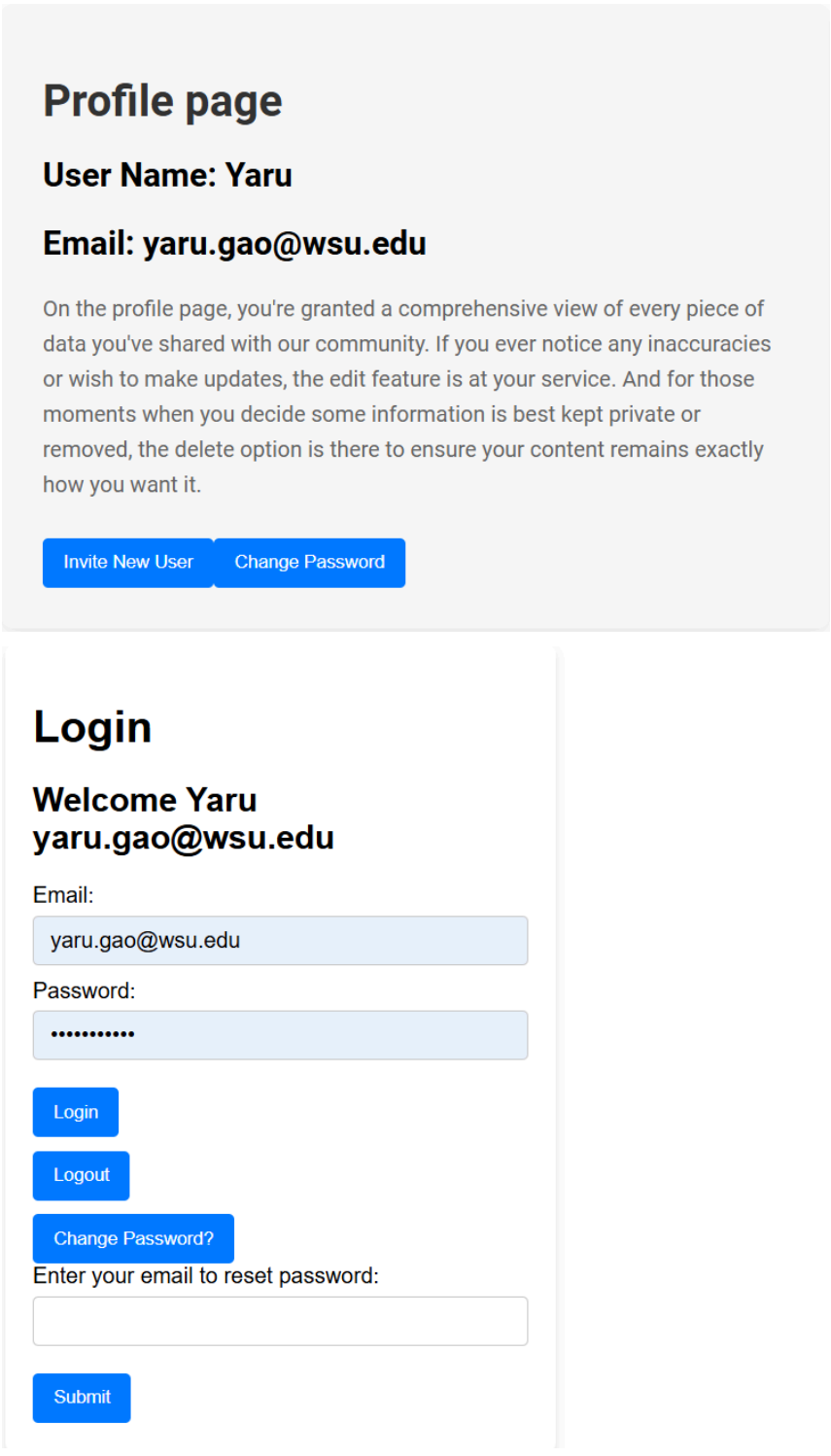


Appendix G: Mockup diagram of the "Learn More" window, showing the edit option for modifying geospatial data.

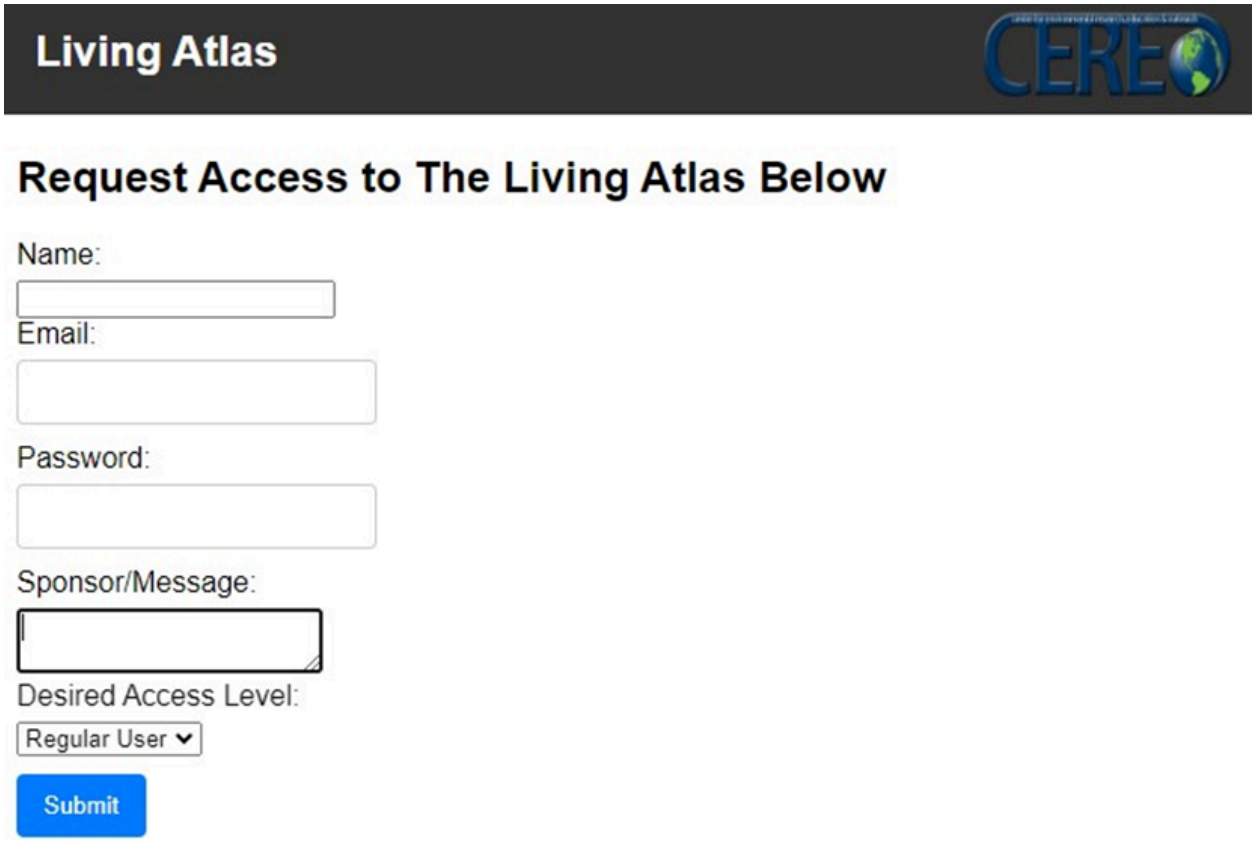
Appendix H: Mockup diagram of the edit functionality within the "Learn More" window.



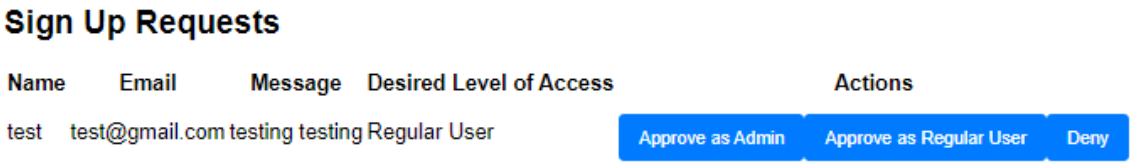
Appendix I.1 and H.2: Mockup diagram of the user center, depicting the password reset process.



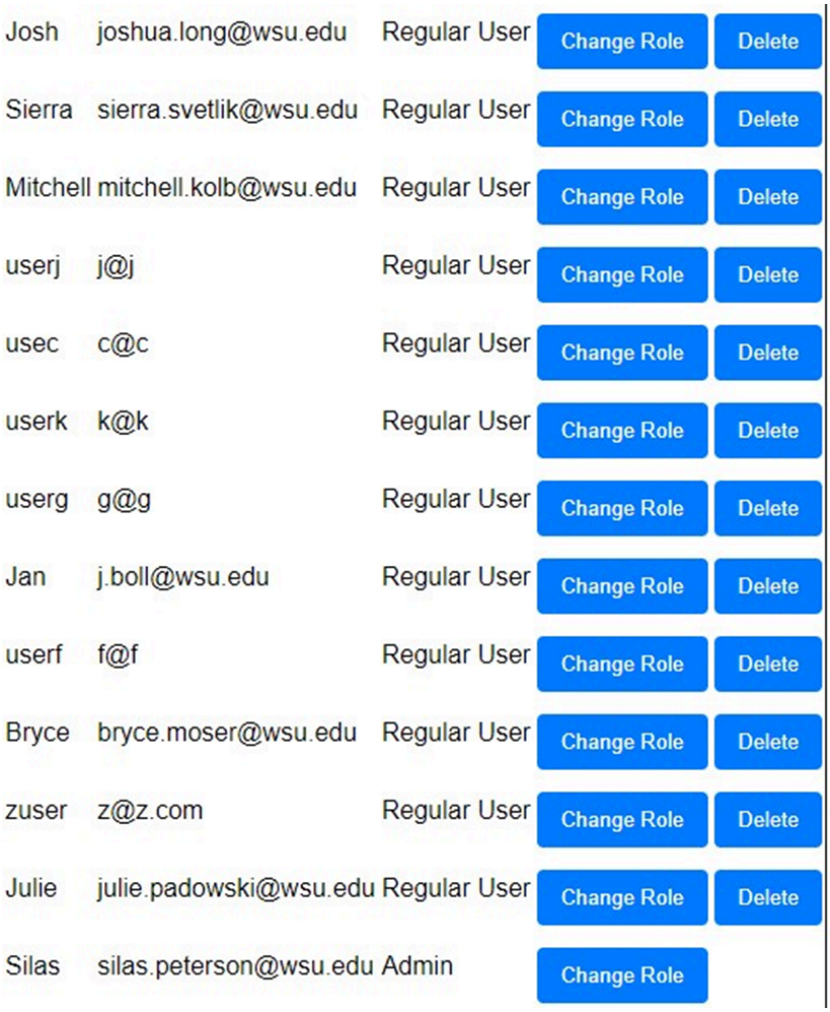
Appendix J: Mockup diagram of the registration page, showing the user account request form.



Appendix K: Mockup diagram of the administrator panel, illustrating user request approvals.



Appendix L: Mockup diagram of the administrator dashboard, showing user management features for modifying access levels and deleting accounts.


Josh joshua.long@wsu.edu Regular User Change Role Delete
Sierra sierra.svetlik@wsu.edu Regular User Change Role Delete
Mitchell mitchell.kolb@wsu.edu Regular User Change Role Delete
userj j@j Regular User Change Role Delete
usec c@c Regular User Change Role Delete
userk k@k Regular User Change Role Delete
userg g@g Regular User Change Role Delete
Jan j.boll@wsu.edu Regular User Change Role Delete
userf f@f Regular User Change Role Delete
Bryce bryce.moser@wsu.edu Regular User Change Role Delete
zuser z@z.com Regular User Change Role Delete
Julie julie.padowski@wsu.edu Regular User Change Role Delete
Silas silas.peterson@wsu.edu Admin Change Role