**Module 7: Project Two Readme**

Charles Breuer

Southern New Hampshire University

CS 340: Client/Server Development

June 17, 2023

## About the Project/Project Title

This project is a visual interface developed to bridge the gap between a database of local animal in the Austin, TX region and the end users. It provides a user-friendly platform where users can interact with the database, filter and search for animals based on specific criteria, and visualize the data through interactive charts and maps. The project aims to enhance the user experience and facilitate the exploration and identification of rescue animals, ultimately allowing for well educated decisions on animal rescue team choices.

## Tools

**Full List of Dependencies Utilized:**

## **MongoDB**

## A NoSQL document database that provides scalability, flexibility, and powerful querying capabilities.

**Mongosh**

A command-line interface and interactive JavaScript shell for MongoDB that allows for easy interaction with the database.

**PyMongo**

A Python library that provides tools for working with MongoDB, allowing seamless integration between Python and MongoDB.

**Dash**

A Python framework for building web applications with interactive visualizations, providing a powerful view and controller structure.

**Flask**

A lightweight web framework in Python that provides the foundation for building web applications, serving as the backend for the Dash application.

**Plotly**

A graphing library that enables the creation of interactive, customizable, and visually appealing graphs and visualizations within Dash applications.

**Rationale for tools:**

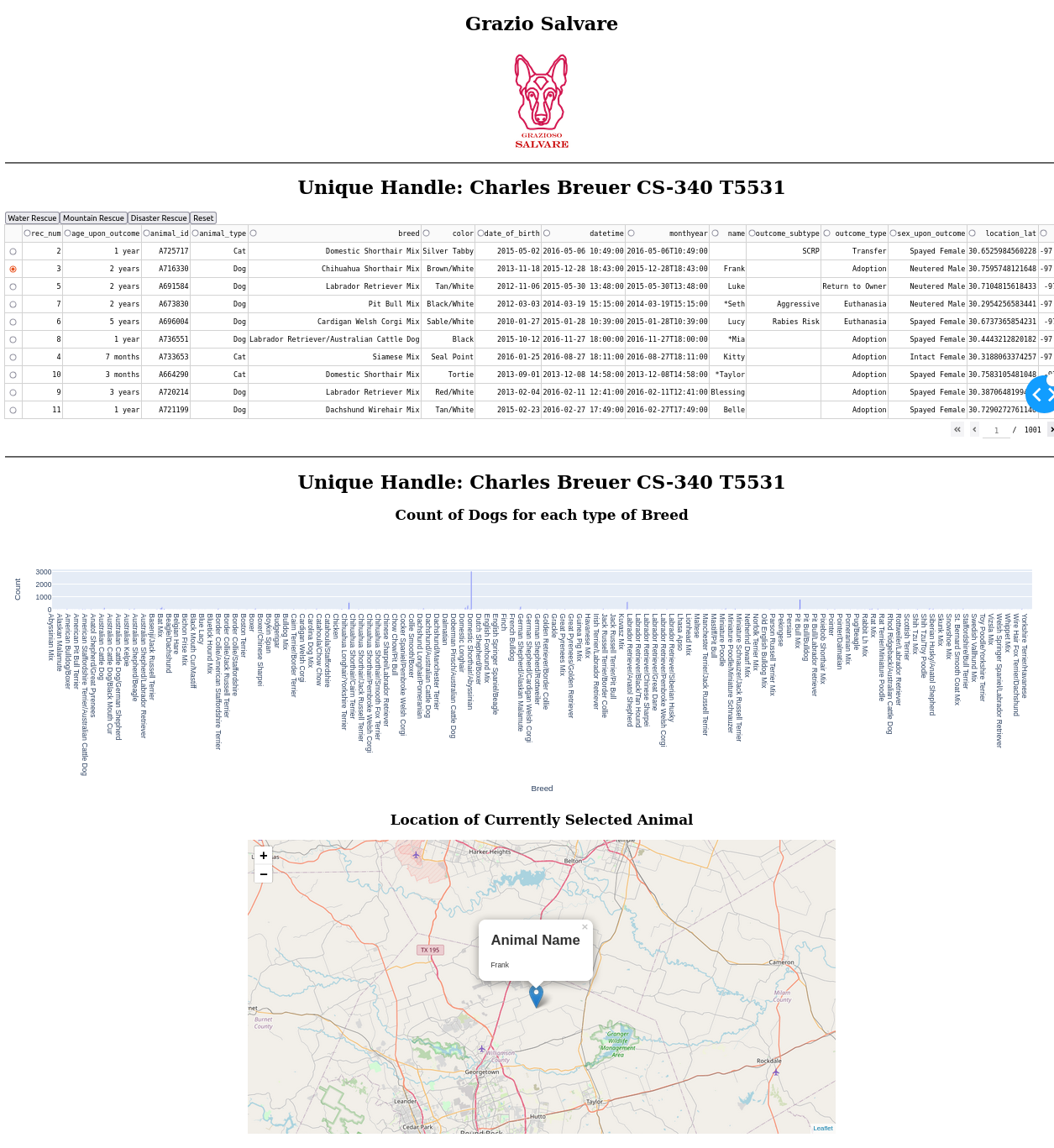
MongoDB was chosen as the model component of the development due to its flexibility, scalability, and compatibility with Python. Its document-based data model, which stores data in a JSON-like format called BSON, aligns well with the .csv file type, making it easy to import and work with the data. Additionally, PyMongo, Dash, Plotly, and Flask, which were used in conjunction with MongoDB, all have native integration with each other, allowing for seamless communication and data flow between the tools.

The Dash framework was chosen for its ability to create reactive content specifically designed for data visualization. Dash provides a straightforward method for creating interactive and dynamic web applications with a focus on data-driven visualizations. It seamlessly integrates with Python libraries like Plotly, allowing for the creation of interactive charts, graphs, and maps. With Dash, developers can easily build responsive and user-friendly interfaces that enable users to interact with and explore the data visually. Its reactive nature ensures that the content automatically updates in response to user actions, providing an engaging and interactive user experience.

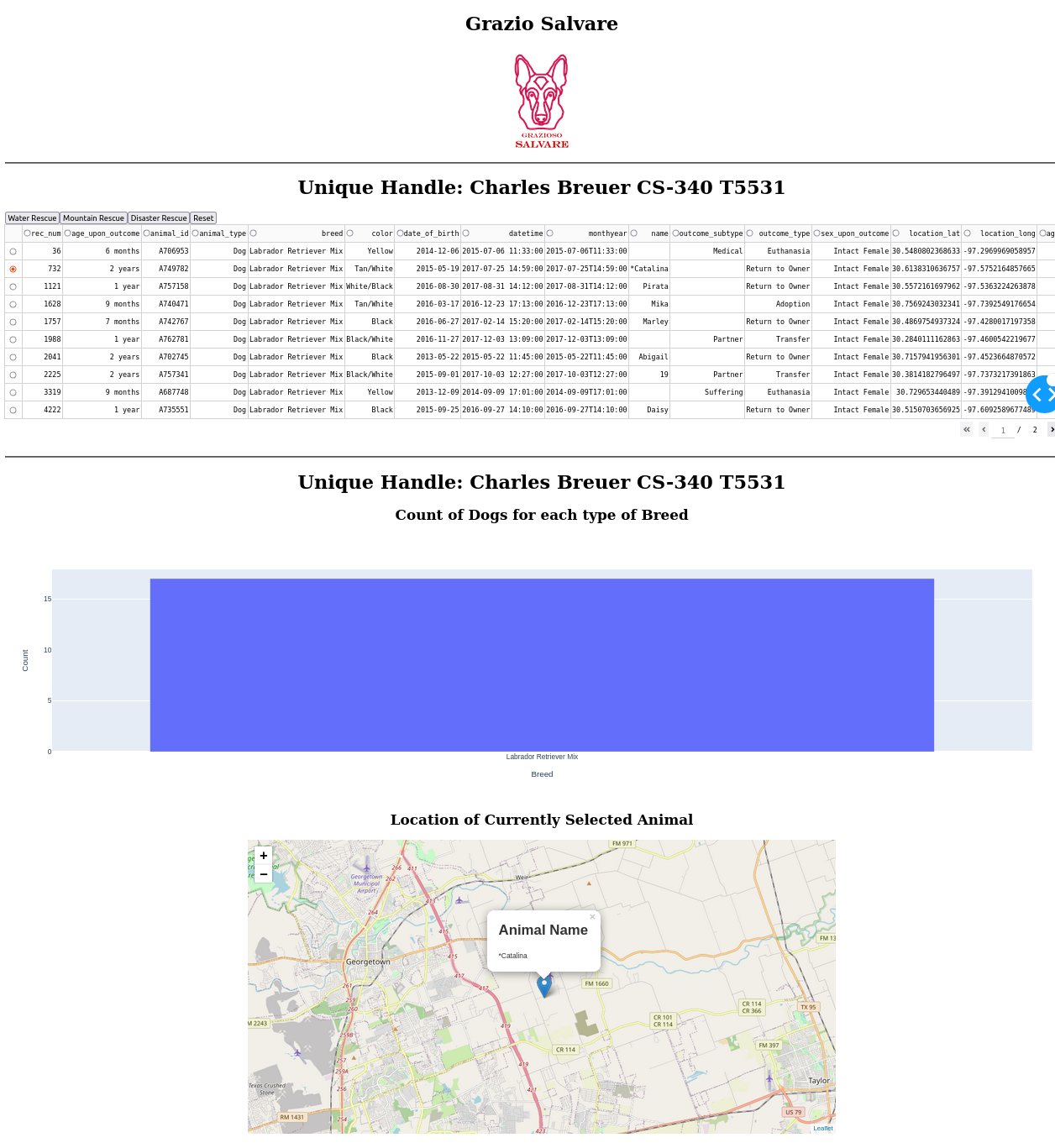
## Usage (Functionality)

Users can interact with the four buttons listed on top of the data-table. These buttons change the filtering state of the database. Changes are reflected in the table, bar graph, and geomap.

Default State: Complete database of animals near Austin, Texas. The bar graph represents the overall distribution of animal breeds without specific filtering. The geomap displays the location of individual animals.



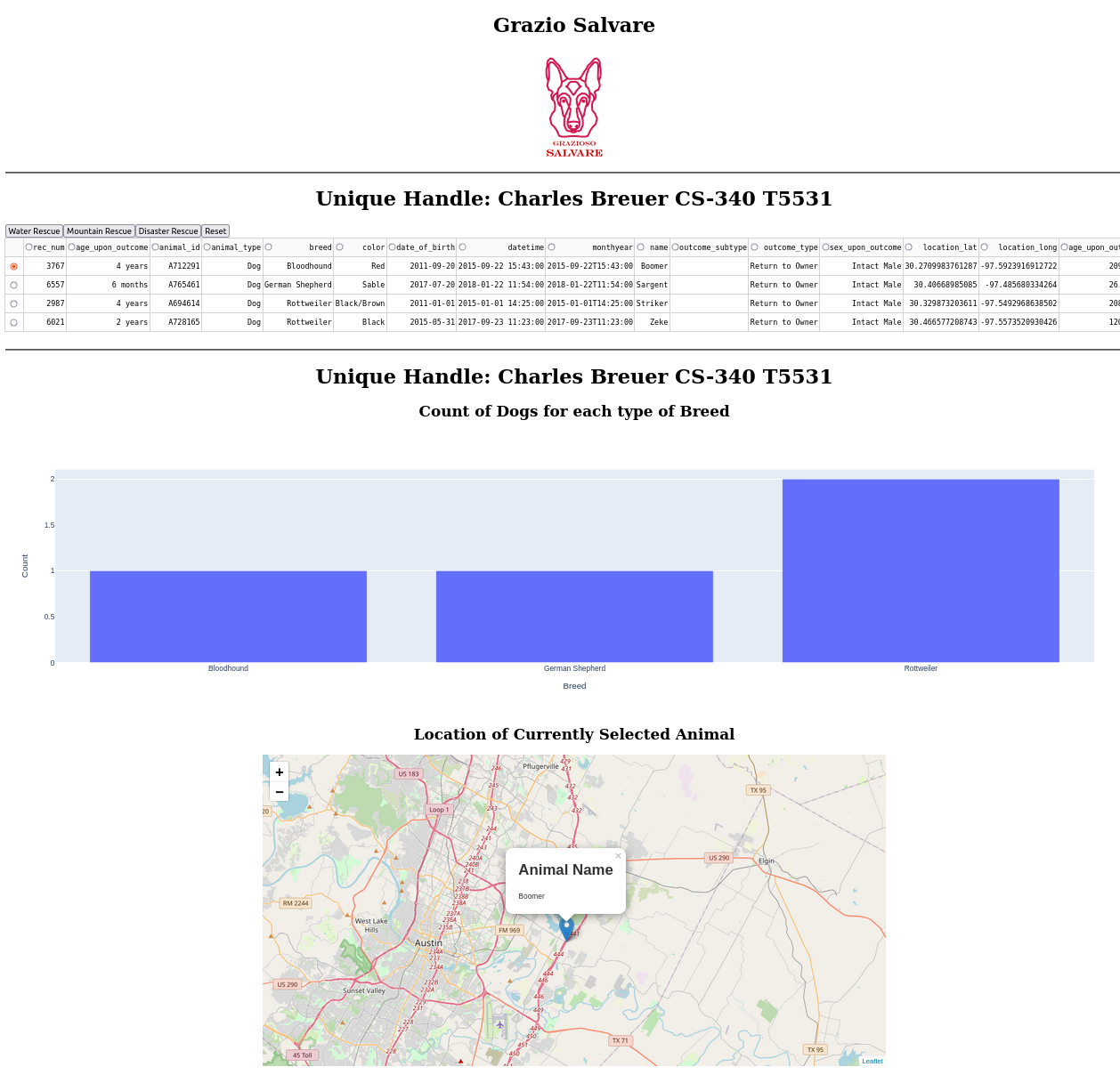
Water Rescue: filter shows dogs with breeds known for water rescues, filtered by availability and age suitability. The bar graph reflects the distribution and count of suitable dogs by specific breeds. The geomap represents individuals within these filter settings.



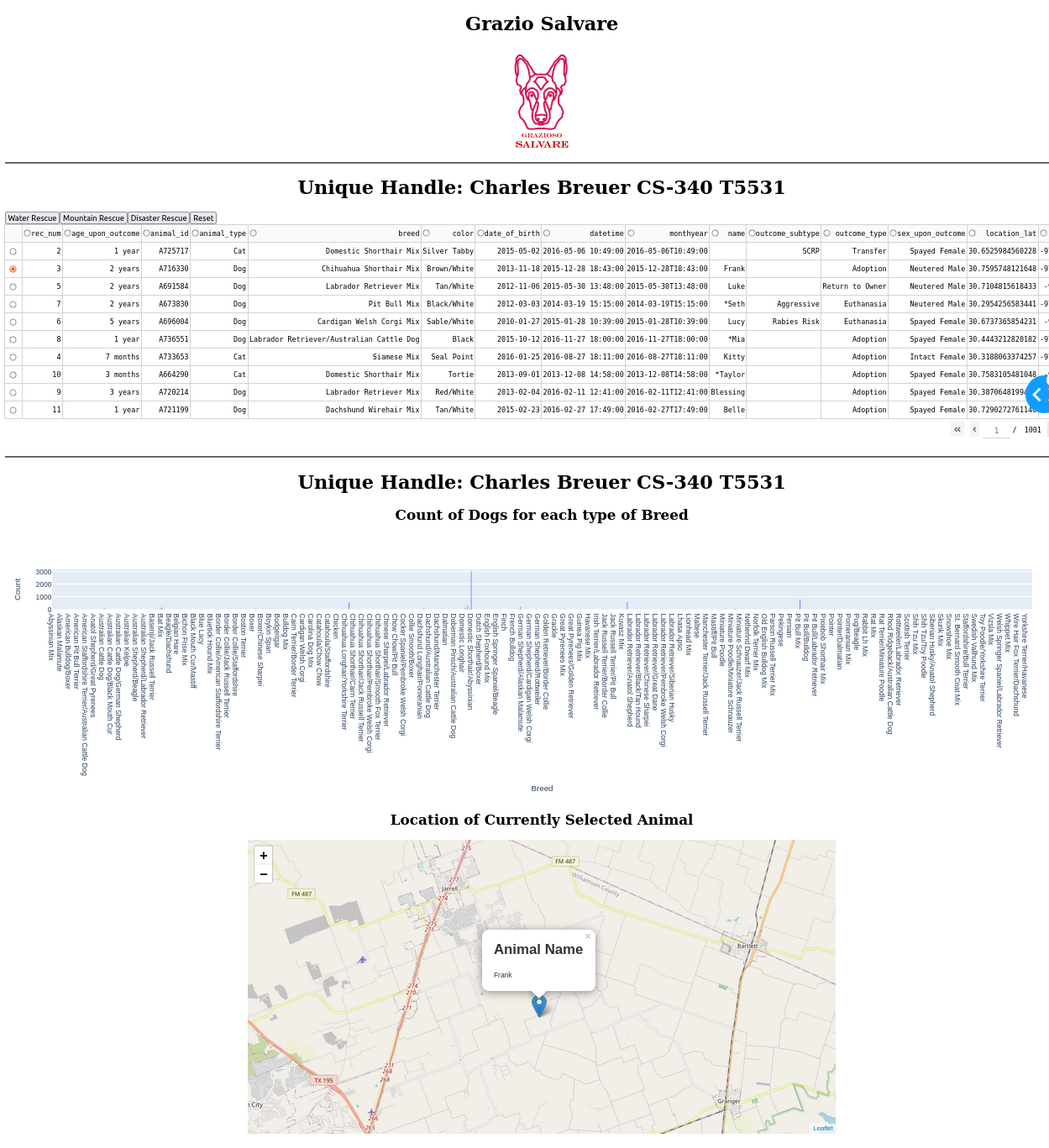
The Mountain: Rescue filter displays dogs with breeds suitable for mountain rescue scenarios, filtered by availability and age suitability. The bar graph and geomap provide insights into the frequency and locations of dog breeds that match this criteria.

### 

Disaster Rescue: filter shows dogs with breeds suitable for major disaster rescue situations, filtered by availability and age suitability. The bar graph and geomap highlight the distribution and concentration of dog breeds that are eligible.



Reset: selecting the Reset option returns the dashboard to the default unfiltered state, clearing any applied filters. This reverts the table, bar graph, and geomap to display the complete database of animals in the local Austin Texas region.



## Project Completion Steps:

**Import Database Files in MongoDB:** This step involved extracting the data from the .csv file and populating the appropriate collections in the MongoDB database.

**Establish Authentication in Database:** This step ensured that only authorized users can access and modify the database, adding an additional layer of security.

**Create pyMongo interface Module:** Create an interface application using pyMongo to provide CRUD (Create, Read, Update, Delete) operations for the AnimalShelter database. This interface allows users to interact with the database and perform operations such as adding new animals, updating their information, retrieving data, and deleting records.

**Establish Dash Server Code and User Interface:** This step involves setting up the server code using Dash, a Python framework for building web applications. The user interface design focuses on creating an intuitive and user-friendly layout for displaying the data from the AnimalShelter database.

**Create Reactive Application through Read Operations and Call-Back Functionality:** This means that the application is set up to update and refresh the displayed data in response to user interactions. Specific read operations are used to filter the database. Dash call-backs allow updated database output to be reflected in the table, bar graph, and geomap.

**Perform continuous testing throughout the development process:** This step involves testing the functionality, performance, and usability of the application at various stages to ensure its reliability and identify any potential issues or bugs.

**Challenges:**

Here is a list of major challenges encountered while developing this application

**Ensuring proper functionality of third-party applications**

**Solution:** Extensive research into the API's documentation was conducted to understand integration requirements and resolve conflicts. This involved studying the documentation thoroughly, exploring examples, and seeking guidance from the API's support resources.

**Hard-to-diagnose bugs impacting multiple layers of the application**

**Solution:** The application was broken down into separate units or modules to narrow down the source of the problem. By systematically isolating and testing individual components, I was able to identify the specific area where the bugs were occurring and focus on resolving them.

**Ensuring accurate filtering results**

**Solution:** Rigorous testing and validation were performed, covering different filtering combinations. Discrepancies were analyzed, and adjustments were made to the filtering logic to improve the accuracy of the results.

**Providing quick and responsive user interactions**

**Solution:** I ensured that callback functions used in the application were optimized for performance. Efforts were made to write efficient code and minimize processing time within callback functions. This approach helped to ensure quick execution and prevent the interpreter from hanging, resulting in a more responsive user interface.

## Contact

Your name: Charles Breuer