# Grazioso Salvare DogFindr README

## About the Project

DogFindr is an application in development for Grazioso Salvare that allows for handling of data from animal shelters to assist in identifying and categorizing available dogs as potential candidates as search and rescue dogs.

## Motivation

DogFindr exists because there is a vast number of canines, and breeds, and their traits and capabilities differ greatly based off of breed and age. Younger dogs typically train more effectively and different breeds perform better in different environments or at different tasks. DogFindr gives the user the capability to look for specific attributes such that the best candidates for the job are presented accurately and timely.

## Getting Started

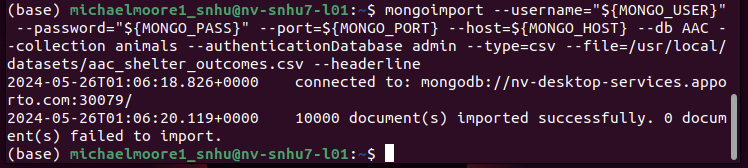
The Mongo database was setup and modified using MongoDB documentation in the following steps:

1. Starting with MongoDB, a database needs to be created and populated for usage with the Python files. Our implementation utilized data from a .csv file, from Austin, Texas animal shelter data, which was imported into MongoDB.
2. From there, our team implemented indexes for the database to improve querying performance, but this is not required.
3. The root user of MongoDB allows for all permissions access to the database in MongoDB, but other users can be created. The team created an additional user to showcase user authentication and permissions controls of users with the Python files accessing the Mongo database.

At this point the Mongo database is setup for use with the Python files. The following steps take place within the Jupyter Notebook:

1. Open both the DogFindr.py file and ProjectTwoDashboard.ipynb file in Jupyter Notebook. From here, one can see the .py file and its contained class for handling interacting with MongoDB and creating, reading, updating, and deleting database entries. The .ipynb file creates the webpage, utilizing JupyterDash, with Dash functionalities using HTML to create the web page.
2. In Jupyter, run the .ipynb file which starts the dashboard on the web page specified on the host and port number in the .py file.
3. From here, the program is ready to use. It initially provides an unfiltered list, that can be manually filtered, sorted, and paged through. For Grazioso, specific filter tabs are implemented to allow searching for different types of rescue animals corresponding to the requested attributes supplied via Grazioso.

Refer below to the software requirements of DogFindr



The above screenshot showcases the mongo import of the csv file of animal shelter data.



The above screenshot shows me changing my Mongo username and password to that of the account I created. I then access the mongo shell as that user and return connection status, which shows I am authenticated as aacuser from the db admin and I have the ability to read and write on the database AAC. This is after I created the new user in the admin database using mongo commands.

## Installation

DogFindr was created and ran on an Ubuntu system. MongoDB was installed on the system along with Jupyter Notebook to handle the Python development and execution. MongoDB was chosen as the database platform as its ease of use with Python via PyMongo. PyMongo allows for access of a Mongo database via Python code that is converted into MongoDB commands. Jupyter Notebook was utilized for the .ipynb file type, which allows for easy code generation and manipulation for developing and running the Dash framework for the web application. The Dash framework was chosen for its ease of setup and modification, in Python, for the basic web application of DogFindr. In its current state, these are the requirements to utilize DogFindr:

* MongoDB installed using the instructions from the MongoDB website (multi-platform support).

<https://www.mongodb.com/>.

* PyMongo is a distribution that contains tools for integrating Python code to interact with MongoDB. This allows for greater versatility for working with the database vice having to manually enter information via a terminal window. DogFindr requires this dependency.

https://pymongo.readthedocs.io/en/stable/

* Jupyter Notebook, or other Python handling application, either utilized in browser or installed using the instructions from the Jupyter website (multi-platform support).

<https://jupyter.org/>

* Dash Python framework was utilized for setting up the Python based web application. It allowed for styling and implementing the HTML web page, utilizing easier Python syntax.

<https://dash.plotly.com/>

## Usage

### Code Example

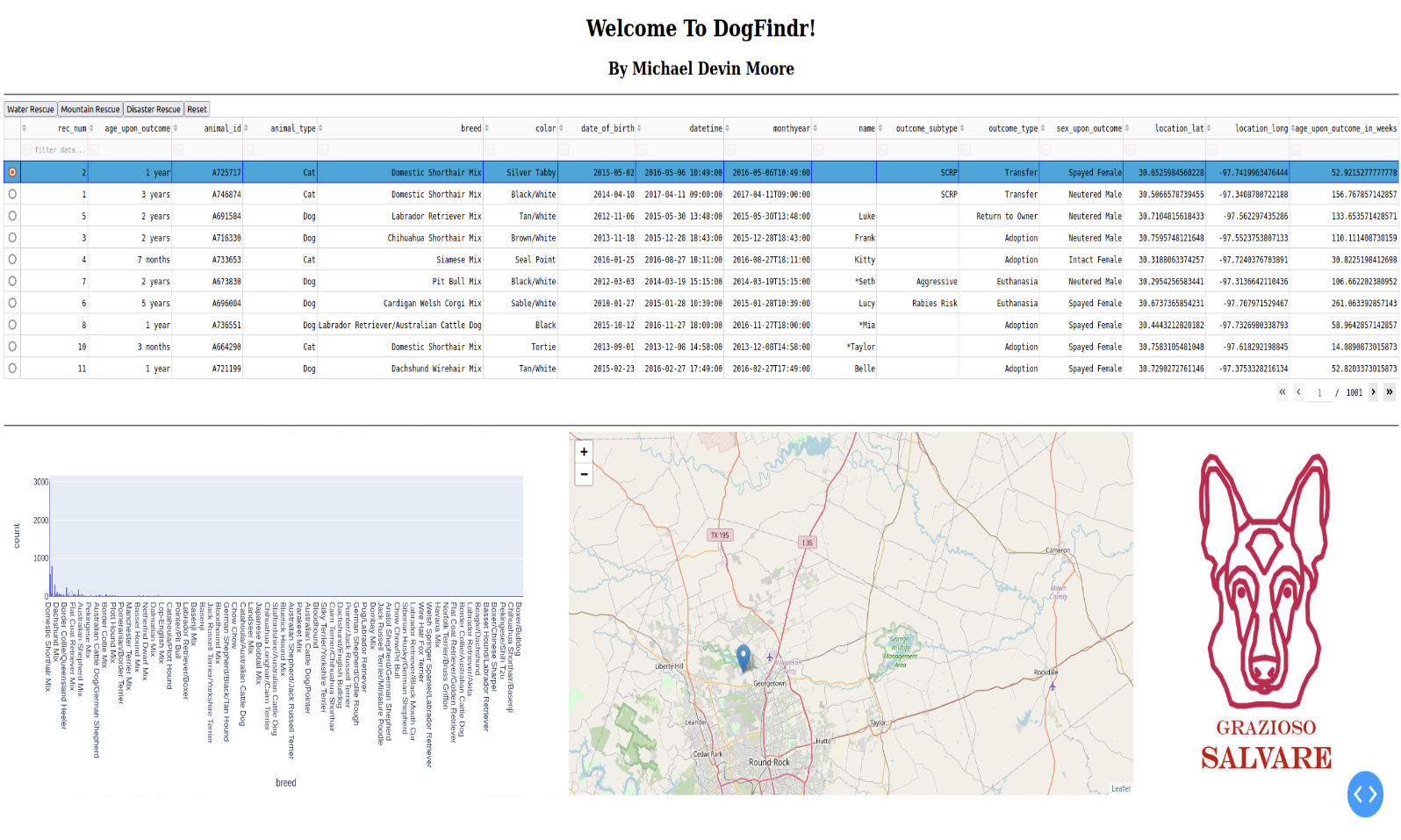
The .py file allows the user to add, search for, update animal entries, and delete animals from the database. The create method covers the adding of animals. This is accomplished by intaking key and value pairs, in proper syntax, to be added as entries into the database. The read method covers searching for database entries via key and value pairs as well. The update method requires two inputs. One is the key and value pair of which to search and match to while the second is the key value pair of which to update any matches to. The delete method takes a key value pair and deletes any animal entries with the data that matches the searched values.

For this implementation of my Python CRUD module only the read method is utilized in making of the DogFindr dashboard. The .ipynb file exists to access the MongoDB via the authentication portion of the .py file. From there, the data within the database is loaded and is presented to the user in multiple ways on a webpage created as a Dash web application. A manually filterable table exists that allows for filtering of every single attribute that belongs to each animal. This table displays 10 results at a time and will generate pages for any remaining results over 10 entries. Buttons exist on the side to select animals, which updates the map portion of the page and highlights the selected animal in the table. As mentioned, preset filters, in the form of buttons or tabs, are available for filtering specific categories of dogs that meet criteria for being different kinds of rescue animals. A geolocation map, similar to Google Maps, is generated to show the location of animals within the database. Lastly, a histogram is generated that shows each type of dog, and how many, exist when each of the preset filters are active. The Grazioso Salvare logo is present in the bottom right corner, which when clicked routes the user to the SNHU homepage.

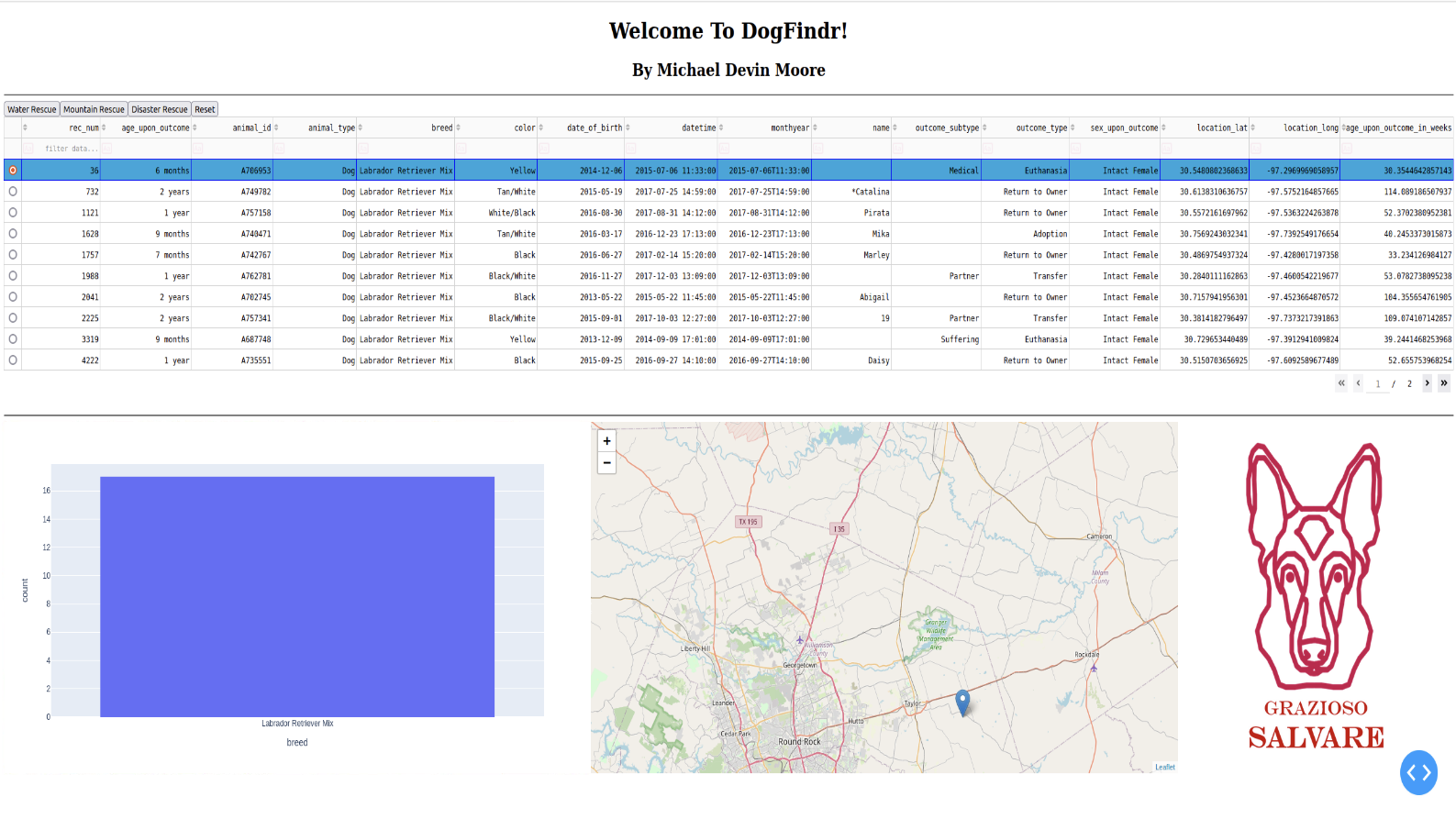
### Tests

The dashboard functions as exactly mentioned. Grazioso specifically requested images of the homepage being loaded, the filtering of dogs in accordance with their criteria, and the main homepage again when the filters are reset. Below will showcase the possible uses of the dashboard and all of its main features.

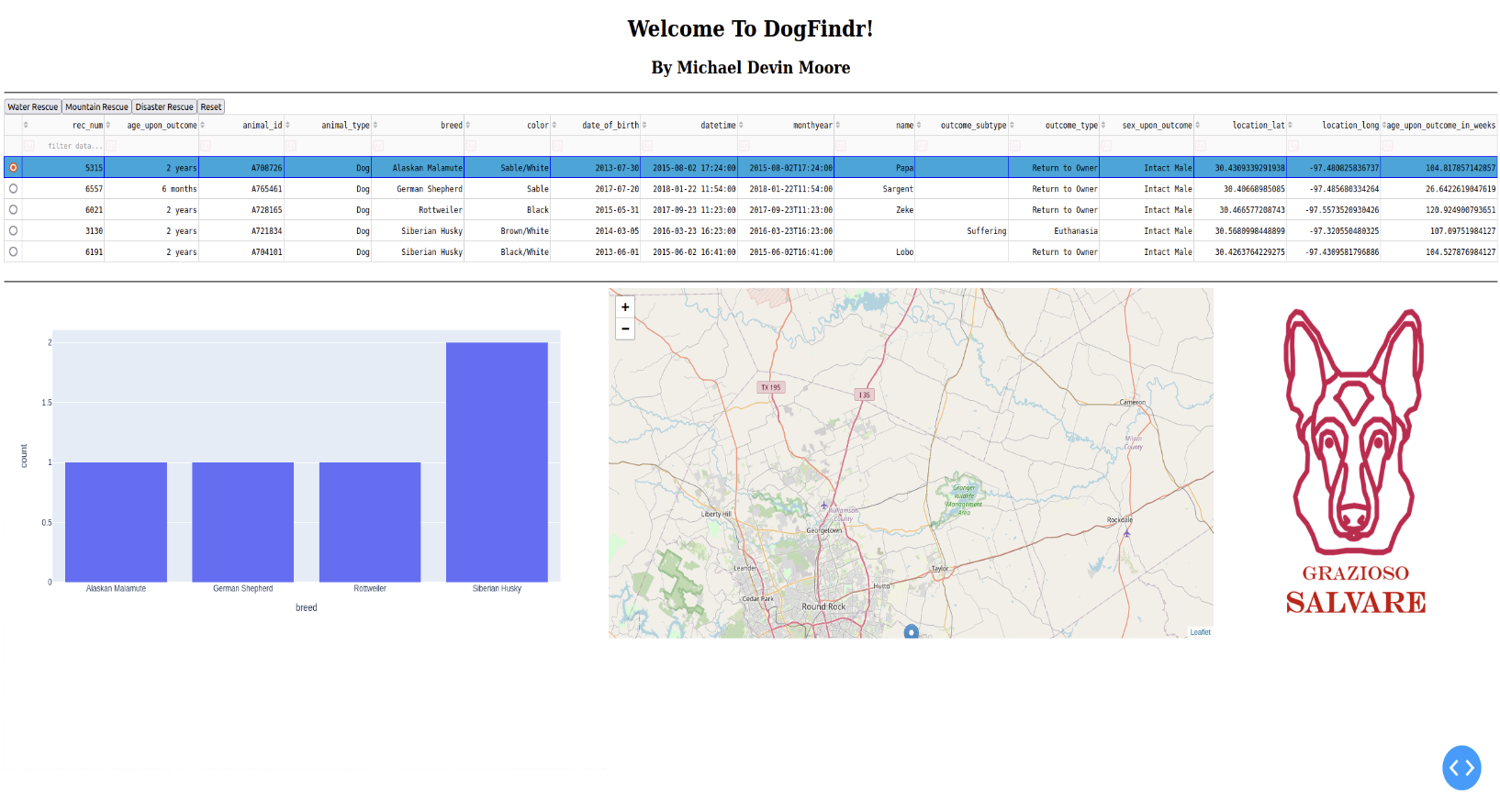
### Screenshots



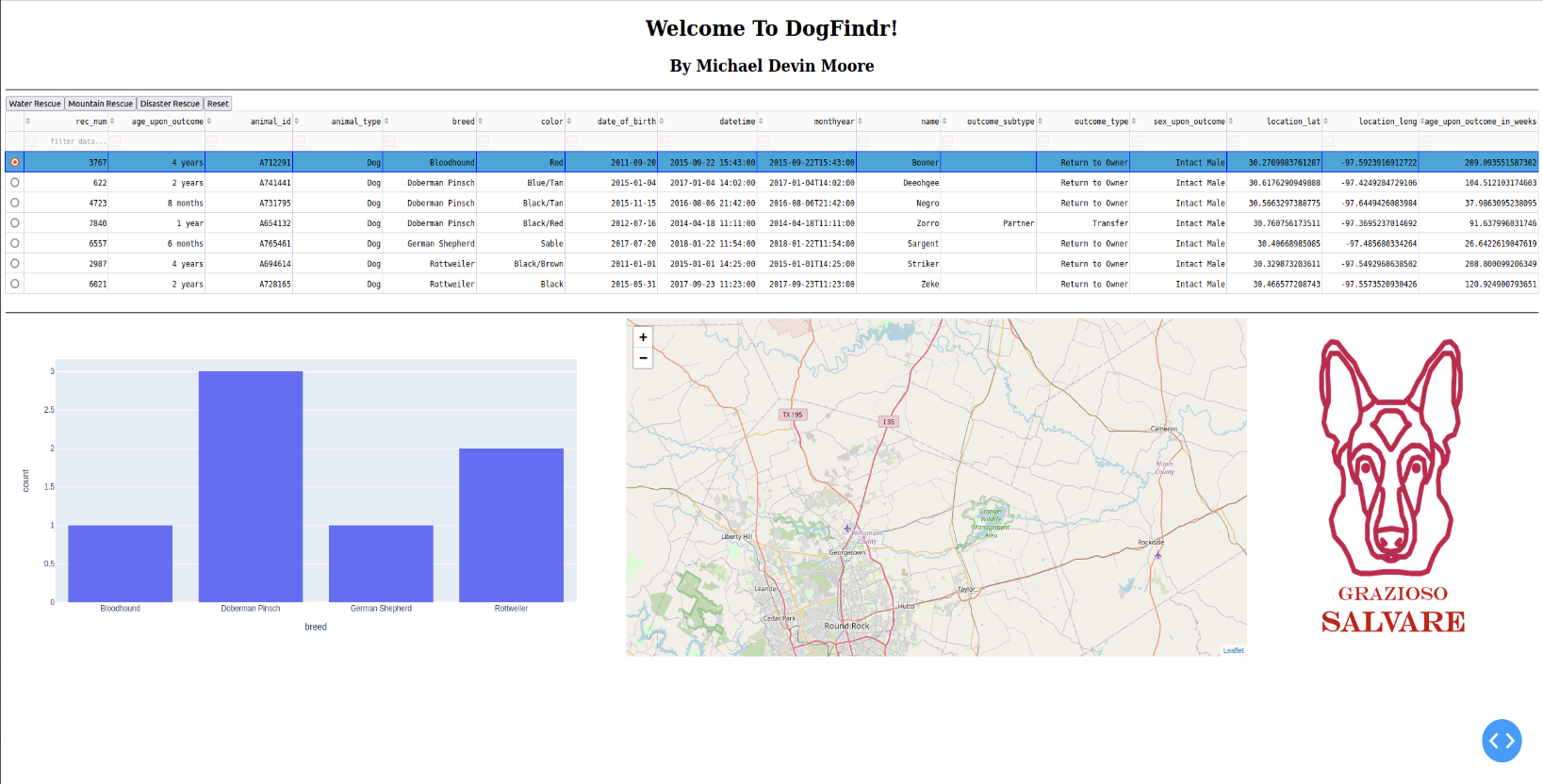
This is a screenshot of the web page upon running of the .ipynb file. The above screenshot showcases that the MongoDB was accessed via the CRUD module and the database was read. The read information was populated into the data table with the histogram generated from the data table showing as much information as it could in the space provided. The map shows the location of the selected animal from the data table, with the selected animals row highlighted in blue. The filter buttons are seen in the top left.



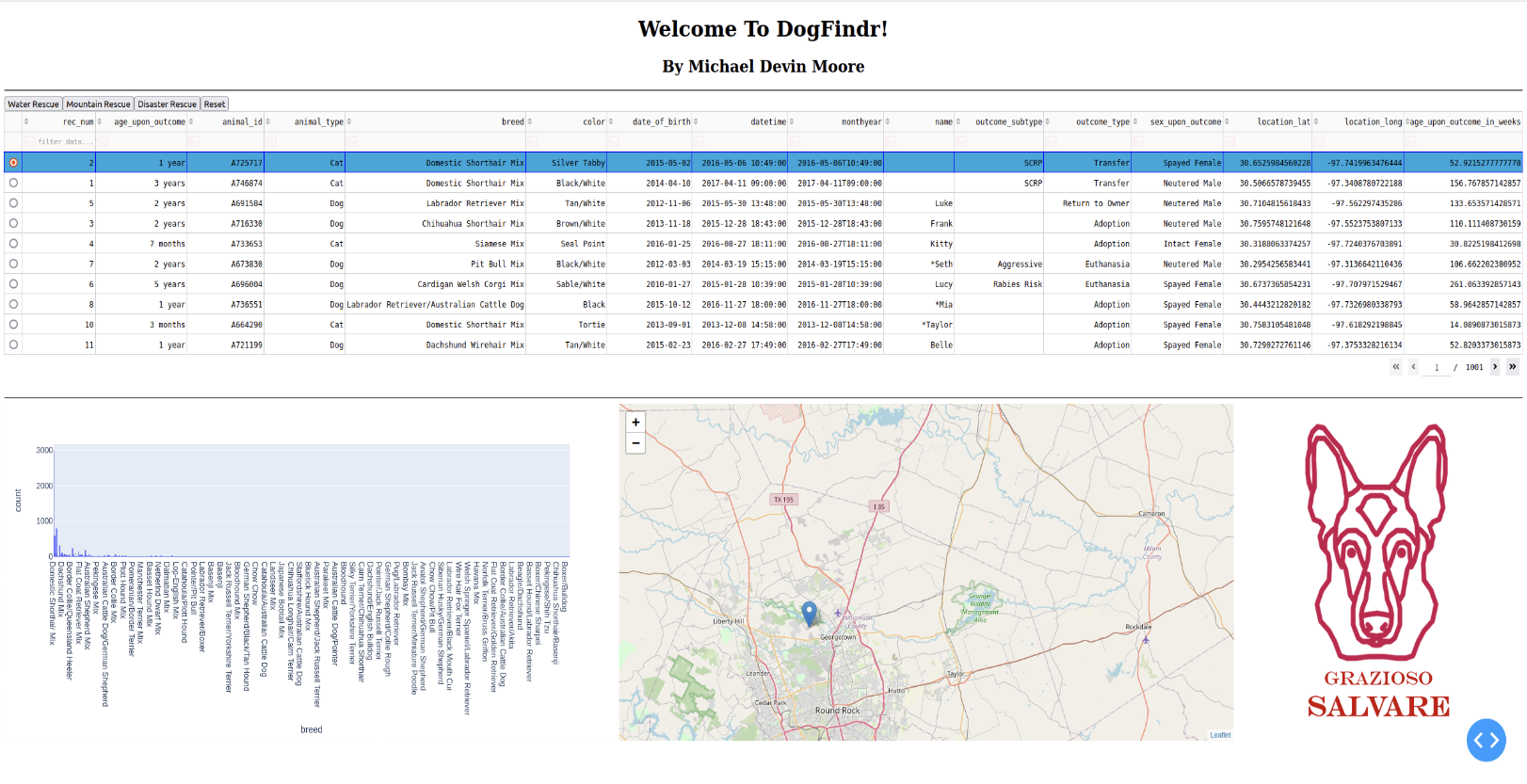
This screenshot is the result of the user interacting with the ‘Water Rescue’ filter option in the top left. It shows all of the dogs that meet the Grazioso criteria of being a water rescue animal. The histogram shows a count of 17 Labrador Retriever Mix dogs. No other breeds met the criteria specified. The data table can show the remaining 7 entries on page two of the table. Additionally, the map updated to reflect the now selected first animal in the data table.



This screenshot shows the result of having the ‘Mountain Rescue’ filter applied to the dataset. The data table reflects all dogs meeting the requirements for mountain rescue dogs. There are 5 entries in total, which are reflected in the histogram by breed and count. There is only one page of the data table available, thus no page selection is available. The now selected animal in the table is reflected on the map as before.



This screenshot shows the result of filtering for ‘Disaster Rescue’ animals. The data table shows the 7 dogs that meet the disaster rescue criteria and they are shown in the histogram as before. The new selected animal is not seeable in the portion of the map that was captured in the shot, but that shows the marker for said animal did move again after filtering.



The shot comes from the ‘Reset’ button removing all preset filters from the data set. The image closely mimics the first shot showing the initialization of the web page.

**Development Reflection**

Development of this program started with Project 1. Project 1 consisted of creating PyMongo methods to authenticate and access a MongoDB database. It also called for the ability to create, read, update, and delete entries within the database. The creation of the Python CRUD module allows for modularity of the module that can be easily implemented in other applications and uses. The final result of Grazioso Salvare consisted of making a client-side web application to interact with the server-side database handling via the Python CRUD module. This web application accesses and showcases database information to the user in a functional and intuitive manner meeting the filter options requested by Grazioso.

The learning experience of this entire project was the usage of multiple frameworks and languages that were all pulled together with MongoDB, PyMongo, Jupyter Notebook, and Dash. Each of these was new to me and required extensive documentation review along with viewing other examples of similar works to fully understand the various methods that are unique to each and their associated languages they work in. I had to do a lot of troubleshooting various errors, especially when it came to Dash and its HTML functionalities. It was not a super intuitive framework for me, but I gained proficiency in it and quickly turned my web application into the useful display of information it is now.

## Roadmap/Features

The current version of DogFindr consists of a MongoDB database that handles user authentication and data storage. It also consists of a Python file which enables a user to create, search, update, and delete database entries. The accompanying .ipynb file allows for accessing the database via the .py file to display data in a tidy and convenient manner for finding rescue animals for Grazioso Salvare via a web application. Future versions of DogFindR will include:

* Continued support with functionality shifts in accordance with Grazioso Salvare requests.

## Contact

Michael Devin Moore

Lead Developer of Grazioso Salvare DogFindr Project

Software Engineer for Global Rain