Dylan Kusick

Professor Paruchuri

CS-340

November 26th, 2023

# CS 340 README

## About the CRUD Python Module

In this project I have worked to create a CRUD Python Module that will allow a user to insert, find, modify, and delete entries in a Mongo Database. This has been developed for Grazioso Salvare, a company that operates animal shelters around Austin Texas. Grazioso Salvare has asked Global Rain to develop a Python module that can work with their existing animal shelter data to identify and categorize dogs.

## Motivation

This project exists to make interacting with the Mongo Database easier. This project is meant to be portable, meaning it can be used for a variety of different database entries. Python is a very user friendly, versatile, and flexible programming language. To expand what Python is capable of, this application will be able to connect Python code to MongoDB. This will ensure users with backgrounds in different languages can easily become connected to MongoDB, allowing for faster development with a variety of different applications. Users will be able to use predefined functions to create, read, update, and delete database entries not only for the project for Grazioso Salvare, but also for many other projects as the code will be provided for users to use and modify to their own liking.

## Getting Started

To get started, you want to ensure you have a Mongo database with users set up. You must then upload the CSV file with the data you are working with into the database using the import tool. (Screenshot in screenshots section) If you have your database user setup, you will be able to go through the login process using the Linux and mongo shells. (Screenshot in screenshots section) Next, call the AnimalShelter class and set it up with any variable to initialize it. Use key/value pairs when working with information in the database, this goes for creating, reading, updating, and deleting data.

I encountered issues when developing the create and read functions due to minor syntax errors and simple misunderstandings. Most of the code needed was included with the create step, which helped with the creation of the code needed for the read function. The code was created for the connection to the database but even after updating the variables, calling the function in the test file was not allowing me to access the database as an authorized user, even when using admin credentials. Pinpointing the errors in the database setup and testing script proved to be very difficult but worked out in the end.

## Installation

This installation requires that you have a Mongo database set up, as well as users set up. Here are some links that go over importing and setting up users in MongoDB:

<https://www.mongodb.com/docs/manual/core/authorization/>

<https://www.mongodb.com/docs/database-tools/mongoimport/>   
<https://www.mongodb.com/developer/products/mongodb/mongoimport-guide/>

MongoDB was used as the model component of the development because of its ability to work efficiently with multiple data types at a large scale.

To use this software, you will need a Python IDE, examples include PyCharm, Jupyter Notebook, and Spyder.

IDE Downloads:

PyCharm: <https://www.jetbrains.com/pycharm/download/?section=windows>

Jupyter Notebook: <https://jupyter.org/install>

Spyder: <https://www.spyder-ide.org/>

Once you have your IDE installed and opened, you will need to create an empty Python file. Once this has been done, the MongoClient needs to be imported so that your Python application can interact with the Mongo Database. The AnimalShelter class also needs to be imported as “from AnimalShelter import AnimalShelter” once you are in your IDE. When the IDE is setup, and your class has been imported, you must create an instance of the AnimalShelter class. Once the class has been initialized, you will be able to use the create, read, update, and delete functions. Look below at the code usage as well as the tests for more clarification on these functions.

The Dash framework has been used to provide the view and controller structure for the web application. It allows this application to use the same data for the front and backend, meaning that there is no need to have unnecessary amounts of code that could be simpllified. When testing this code in Jupyter Notebook, I was able to easily integrate dash imports to work with the project in this IDE.

## Usage

### Code Example

from bson.objectid import ObjectId

from AnimalShelter import AnimalShelter

animals = AnimalShelter()

#CREATE

animal\_data = animals.create({

"type":"Dog",

"breed":"Corgi",

"age":4,

"name":"Phil",

"weight":26

})

if animal\_data:

print(f"Entry added: {animal\_data}")

else:

print("Entry failed")

#READ

findEntry = animals.read({"breed":"Corgi"})

for i in findEntry:

print("Entry:", i)

#UPDATE

updateAnimal = animals.update({"breed":"Corgi"}, {"name":"HOSS"})

print("Updated animal")

#DELETE

deleteAnimal = animals.delete({"type":"Dog"})

print(f"Dog deleted: {deleteAnimal}")

### Tests

**You should be able to run tests similar to the code block above by inserting your own data for the animal creation, utilizing any data pairs you wish. For example, you could use the function below to insert a reptile named Greg that is 13 inches tall, and is the color green.**

your\_instance.create({

‘name’:’Greg’

‘height’:’13 inches’

‘color’:’green’

‘type’:’Reptile’

})

**Looking for this entry could be done by using the read function on the instance of the AnimalShelter class:**

your\_instance. read({‘color’:’green’})

**Updating the animal is easy, just use the instance of your AnimalShelter class paired with the update method. This example will update any reptile to have the name “Toasdster”:**

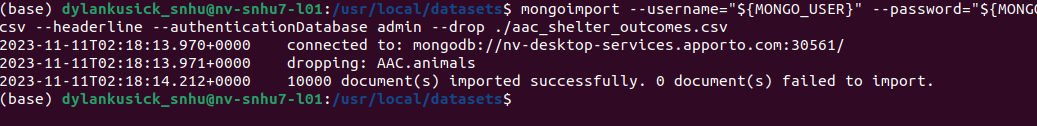
variable = your\_instance.update({"type":"Reptile"}, {"name":"Toadster"})

**Deleting the animal can be done using the instance, in this example, you can delete the animal using the type:**

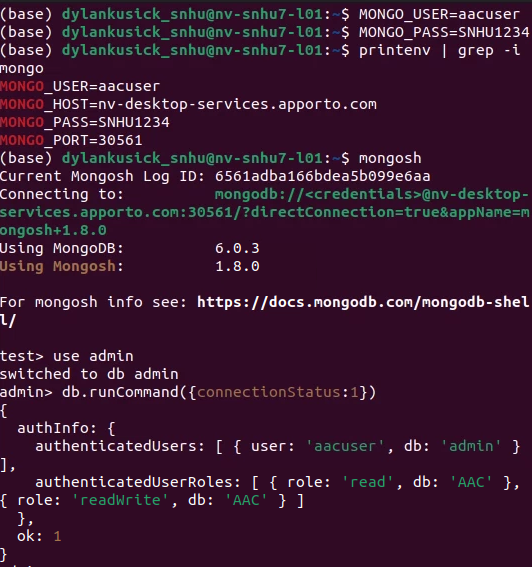
variable = your\_instance.delete({"type":”Reptile"})

### Screenshots

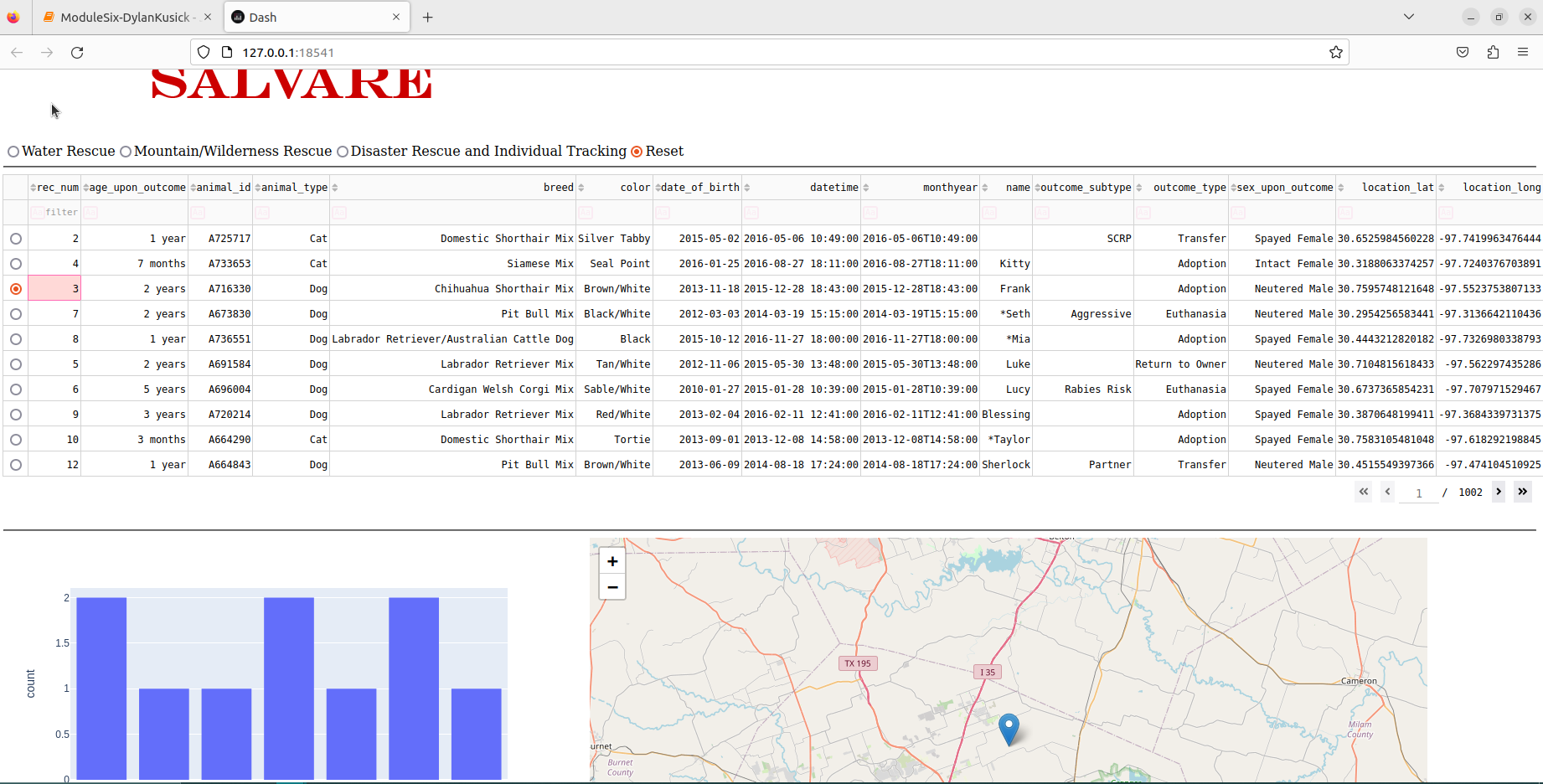
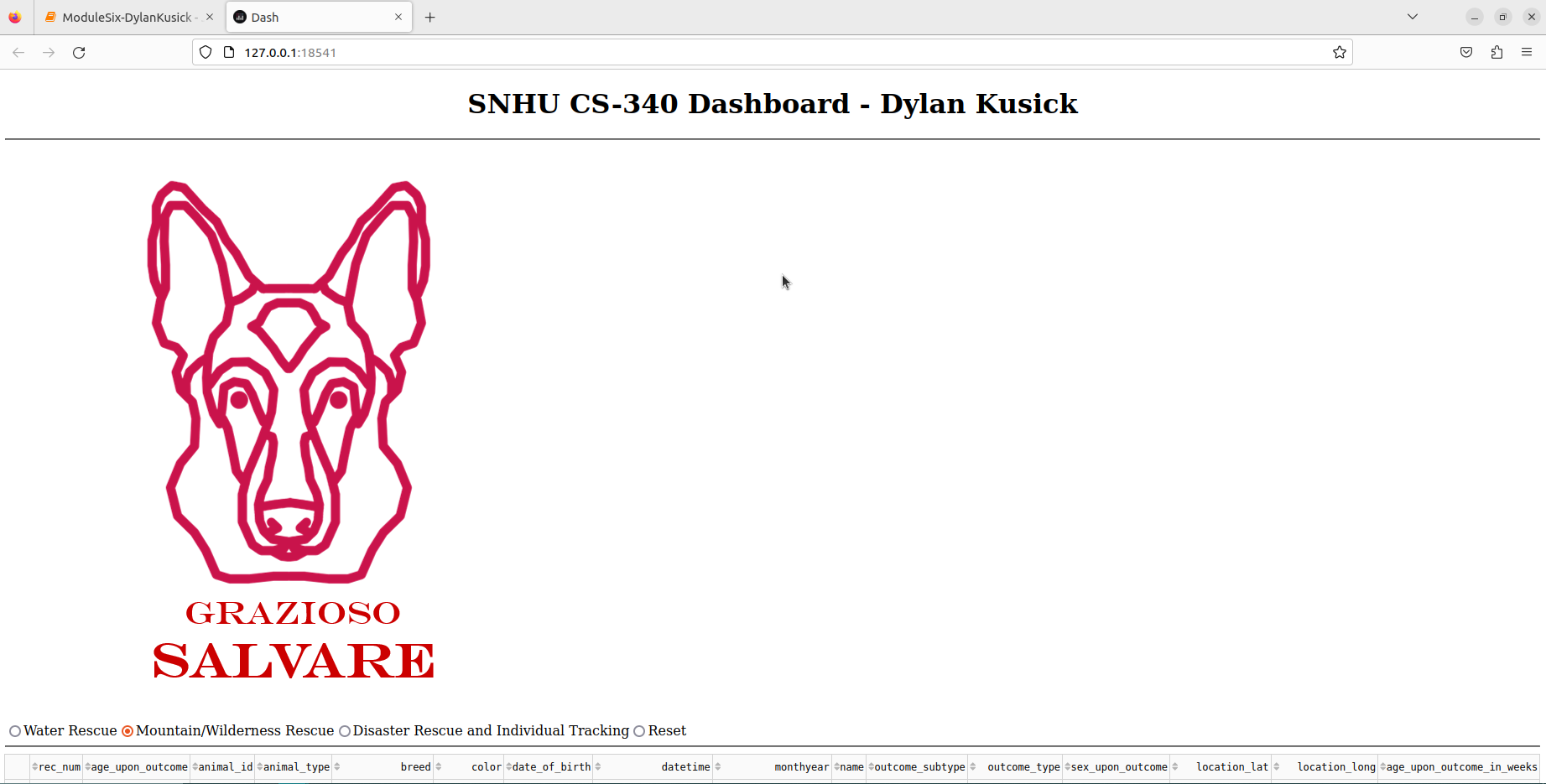
Here is the process of inserting a CSV file into the database:



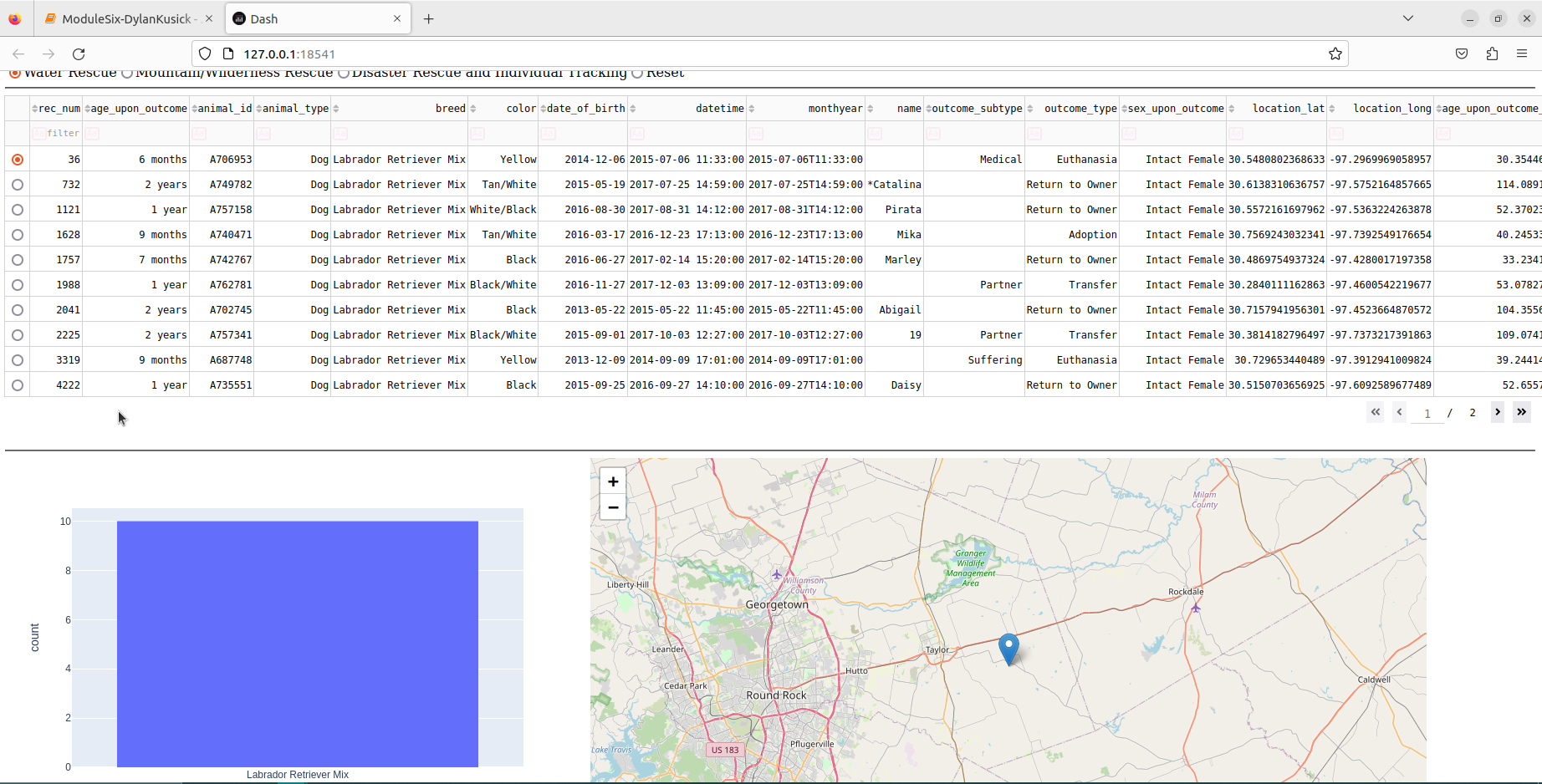
Once your users are setup, this is what the login process to the database will look like:



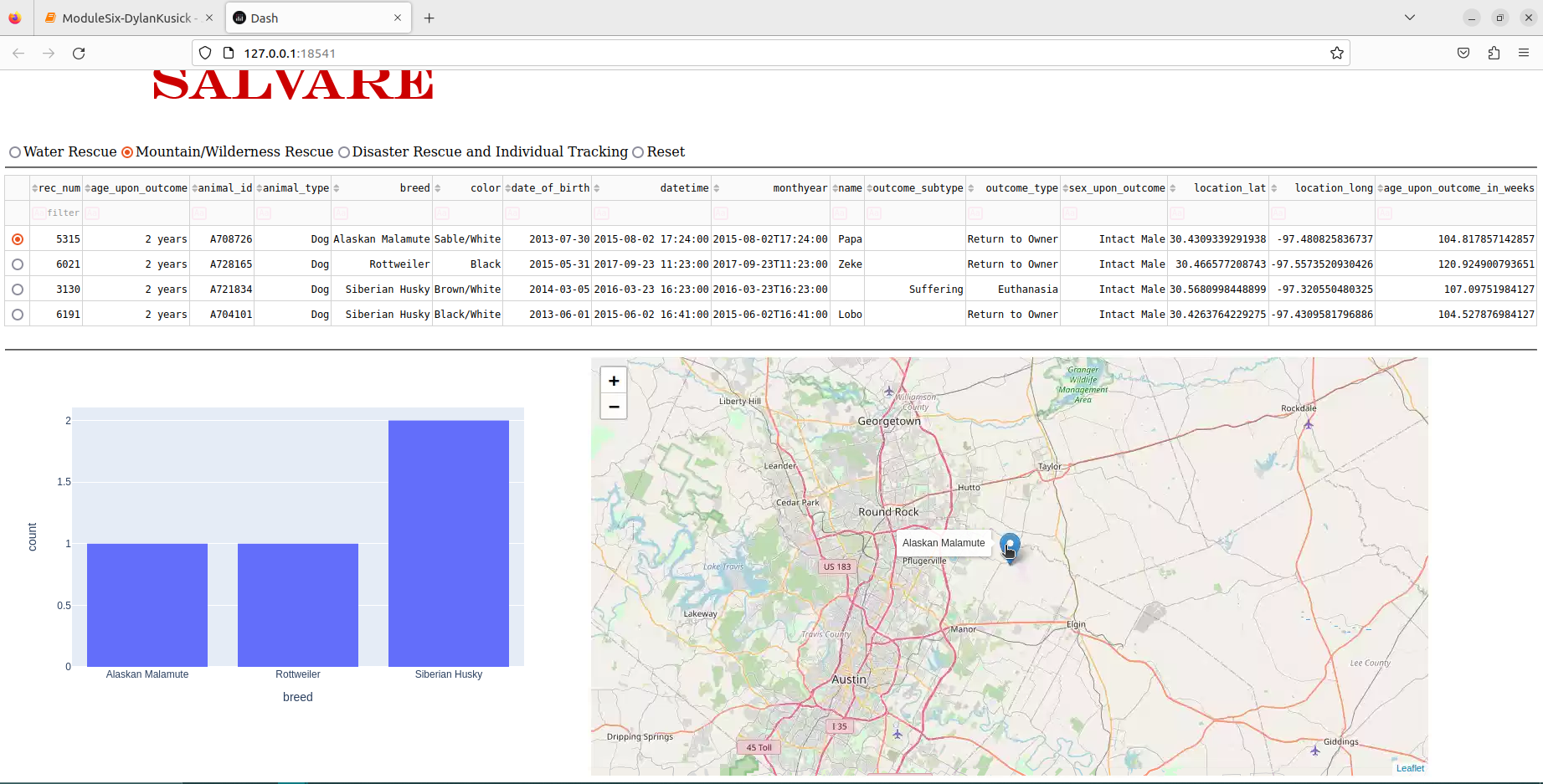
The two screenshots below show the initial launch of the website, as well as the initial state of the data table and interactive map. Clicking the reset button will make the screen look like this initial page:



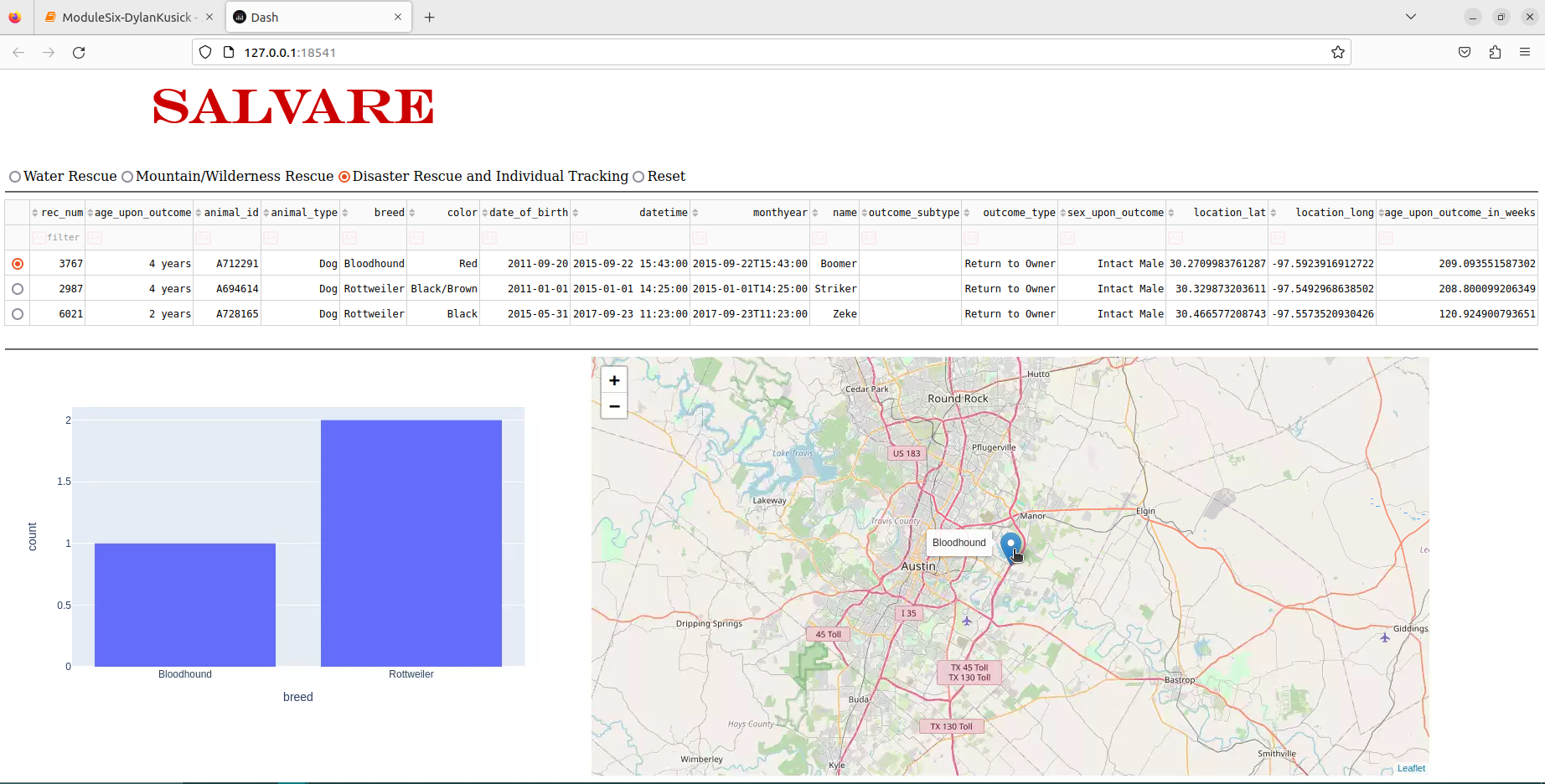
Next up is the picture of the table, histogram, and map reflecting the water rescue filter:



This screenshot shows the mountain/wilderness filter:



This screenshot shows the disaster rescue filter. Once again, changing the data table, histogram, and interactive map:



Below is a screenshot of the test script using all the functions, create, read, update, and delete:



## Roadmap/Features

A known issue for this software was the non-authorization to the database, but this was an error in the database creation. Other issues such as minor syntax errors have been evaluated and fixed. There was also initial errors with the import statements which would not allow me to display the map or data table correctly, this has been fixed for this final project.

A future feature for this project could include the ability to see which entries have been deleted a bit more clearly. This project does delete entries in a database, but the process could be a bit clearer for the user. Any updates that may be made by users on GitHub will expand on what the project is capable of.

If you have any other questions, feel free to reach out to me via GitHub. Happy coding!

Contact

Dylan Kusick