Scott Cain

CS-340 Project 2

6/22/25

**About the Project / Project Title**

This project is an interactive dashboard built to help Grazioso Salvare with identifying animals that fit rescue mission criteria. It connects to a live MongoDB database and pulls data on shelter animals, letting the user filter records, view their locations on a map, and see breed distribution in a pie chart. It uses full CRUD functionality and gives users tools to filter and analyze data that will help determine which animals could be trained for specific rescue needs.

**Motivation**

The purpose of this project was to gain experience working directly with a real-world dataset using MongoDB, and to develop a web dashboard where users could interact with that data. This helped me better understand how to write queries and organize output in a way that works across data tables, maps, and charts. The hands-on development gave me a much better understanding of how database-backed applications are built and how users interact with them to make decisions based on live data.

**Getting Started**

1. Import the aac\_shelter\_outcomes.csv dataset into a MongoDB database called AAC
2. Create indexes on relevant fields (like breed, age, outcome) to speed up filtering
3. Create user authentication with aacuser and an admin account for proper database access
4. Use Jupyter Notebook and JupyterDash to run and view the dashboard locally

**Installation**

This dashboard uses the following tools:

* **MongoDB** – to store and access animal data. Its flexible schema works well for documents with varied fields.
* **Python** – to run CRUD operations and build the dashboard logic.
* **Jupyter Notebook** – to develop and test the app in an interactive environment.
* **pymongo** – to connect Python to MongoDB and perform database queries.
* **Dash / Plotly / Dash Leaflet** – to build the user interface, create the pie chart, and show animal locations on a map.

**Usage**

The dashboard lets the user:

* Apply different filters: Water Rescue, Mountain Rescue, or Disaster Rescue
* View results in a dynamic table
* Click a row in the table to update the map to that animal’s location
* View a pie chart that shows which breeds are most common in each filtered group
* Return to all results with the “Reset” option

The logo at the top links to the SNHU homepage and gives the app a branded, professional feel.

**Why MongoDB**

MongoDB was used because it stores data in flexible documents that are easy to query using Python. That made it ideal for storing animal records that may vary in structure. With pymongo, I was able to build targeted queries that filter by breed, age in weeks, and other fields. It was simple to convert MongoDB results to DataFrames so I could pass that data to Dash components like charts and tables.

**About Dash**

Dash was used to build both the layout and logic of the app. It acts like the view and the controller:

* The view is made up of components like the table, the pie chart, the map, and the filters
* The controller logic is handled through callbacks. When a user selects a filter or a row in the table, Dash automatically updates the other components

It worked really well for this project because it allowed me to build everything in Python without needing to touch any front-end code like JavaScript or HTML.

**Example Code**

Here’s an example query used to find all yellow Labrador Retrievers:

from crud import AnimalShelter shelter = AnimalShelter("aacuser", "Your\_Password") query = { "breed": "Labrador Retriever", "color": "Yellow" } results = shelter.read(query) if results: print("Matches found:") for animal in results: print(animal) else: print("No matches found.")

Screenshots

Below are screenshots from the dashboard showing functionality and proof of implementation:

* Screenshot 1 – Dashboard header, filter buttons, and results table

A screenshot of a computer

AI-generated content may be incorrect.

* Screenshot 2 – Pie chart and map with location marker and popup

A screenshot of a computer

AI-generated content may be incorrect.

* Screenshot 3 – Water Rescue Filter

A screenshot of a computer

AI-generated content may be incorrect.

* Screenshot 4 – Mountain Rescue Filter

A screenshot of a computer

AI-generated content may be incorrect.

* Screenshot 5 – Disaster Rescue Filter

A screenshot of a computer

AI-generated content may be incorrect.

One of the biggest issues was getting the chart to display correctly when there was no data. At first, if a filter didn’t return any records, the app would crash. I fixed that by adding a condition in the callback that displays a message like “No data to display” if the data was empty.

Another issue was that the image file wasn’t loading. I figured out it was a typo — my code was looking for what I thought was 'GraziosoSalvareLogo.png' when the actual file had a space: 'GraziosoSalvare Logo.png'. Once I matched that exactly, the logo loaded fine.