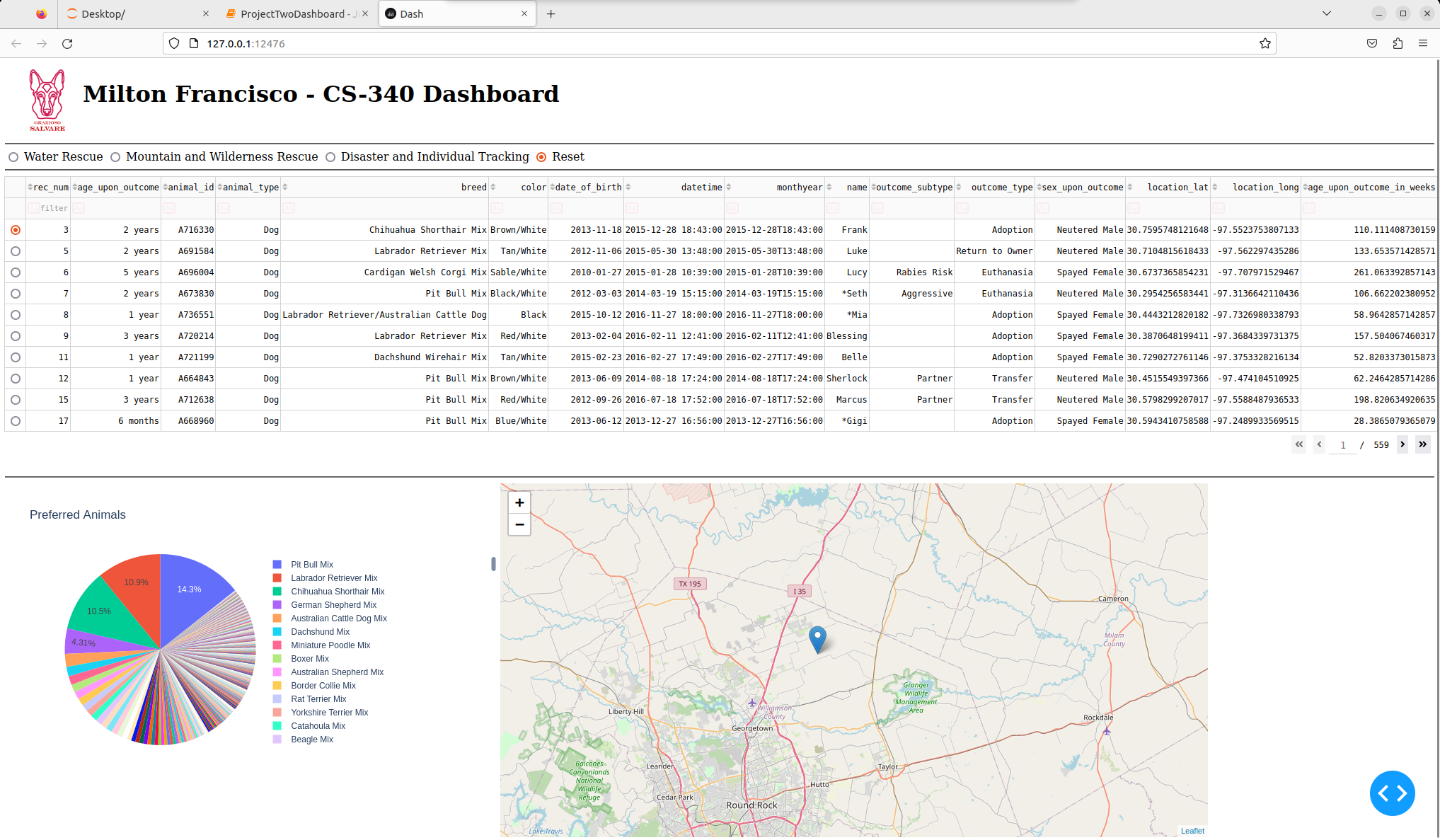
Milton Francisco  
CS – 340  
December 9, 2024

**Web Application Dashboard README**

**Project Overview**

This project involved creating a dashboard application that integrated data visualization, interaction and storage. The required functionality for this project includes dynamically updating a pie chart and map, based on user interaction with a data table.

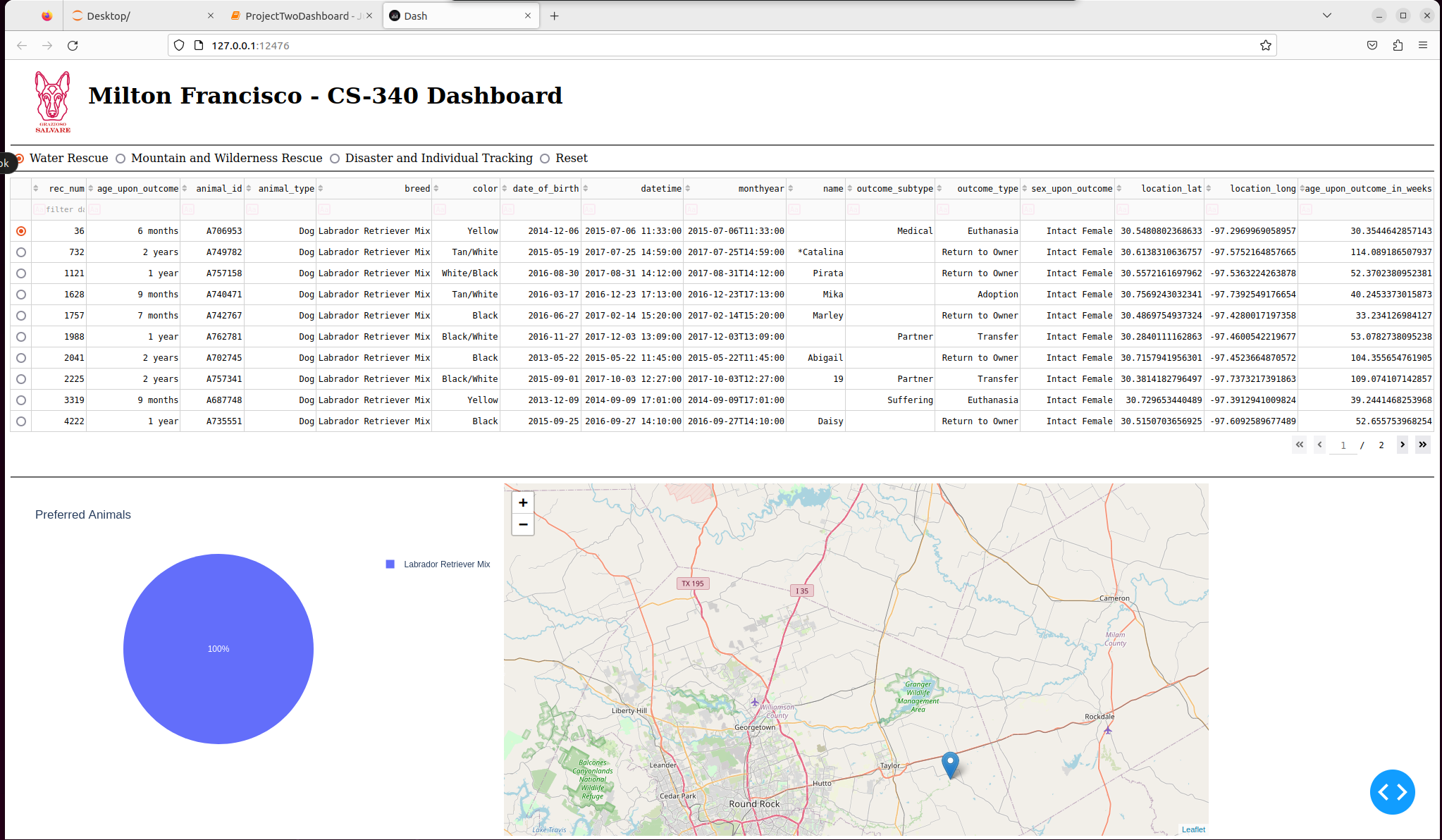
When first deploying the dashboard, the user is met with the following screen:

The app defaults to the ‘Reset’ category and selects the first row that corresponds to an animal. This allows the functionality of the map and the pie chart to be seen as soon as the user opens the app. Additionally, the preferred animal for rescues are dogs, which are displayed in the pie chart on the bottom left. Initially, there are 5590 dogs that are available. This causes the pie chart slices to be quite thin, but each slice shows the breed of the dog when hovered over.

A screenshot of a computer screen

Description automatically generatedOn the top of the screenshot are radio buttons that the user can select. These correspond to types of dogs that are most sought after for specific types of rescues. For example, the Disaster and Individual tracking dogs are shown below:

When the Disaster and Individual Tracking radio button is selected, the data table, pie chart and map are updated with the new information. The map shows the location of the first animal, but the pie chart and data table are reduced in size to the applicable search criteria. This provides a simple way for users to find the applicable dog breeds that are currently contained in the database.

When the user clicks on the water rescue radio button, it shows the applicable dog breeds that are currently held:

This case is a good example of showing how the queries are different from each other. Behind the scenes, the query holds 5 dog breeds, but the database only contains 1 applicable dog breed. In this case, the Labrador Retriever Mix is the only currently housed breed that is suitable for water rescue. This shows how the stored data is converted to useful information for the user.

A screenshot of a computer

Description automatically generatedFinally, the last radio button shows the Mountain and Wilderness rescue:

This final screenshot shows the last of the required functionality. When looking through the screenshots, the starkest difference is in the pie chart. The chart varies in size and is a great indicator of how many dogs are available for a specific task. Upon further inspection, the data table also changes with the data displayed and there is a more subtle difference in the number of pages that are available. In the initial, or ‘Reset’, page there are 559 pages, while the mountain and wilderness rescue only shows 2 pages. This shows how convenient this radio button filter is and will undoubtedly help the users. Additionally, the user can manually filter desired traits using the filter tab at the top of the column. This could allow users to search for a specific color, or a specific breed.

**Tools & Rationale**

**Tools**

* Dash
* MongoDB
* Python
* Jupyter Notebook
* Plotly

**Rationale**

* Dash was used as the framework for this application to aid in creating an interactive and visually appealing dashboard.
* MongoDB was selected to the flexibility with its schema. This makes it suitable for handling diverse and unstructured data formats. Additionally, using pymongo ensured a smooth data retrieval process.
* Python was the driver of this application. The module built allowed the basic CRUD operations to be performed on the database.
* Jupyter Notebook was a simple way to hold the dashboard components and test various features. This was integral during development to help identify any potential issues.
* Plotly was used due to its compatibility with Dash and helped with visualizing the data presented in the table.

**MongoDB**

This was used as the model component of the development due to four main qualities. The flexibility of the schema allowed handling varying structures. The scalability allowed it to handle large datasets efficiently and will be beneficial when the dashboard’s data grows. The ease of integration with libraries like ‘pymongo’ enabled seamless CRUD operations. Finally, query performance supported indexing and aggregation capabilities to aid in quick query processing to ensure the dashboard is responsive.

**Dash** **Framework**

This python-based framework was chosen for this application because it provides multiple components that were beneficial for this application. Dash components like the view layer allowed the frontend of the application to be built in a relatively simple way, and included necessary components such as html components, leaflets, and graphs. Additionally, it provided a way to use callbacks to handle user interactions and dynamically update the page, depending on the user requests. Overall, due to its seamless integration with Python and its libraries, it was an obvious choice when selecting a framework.

**Resources**

* [Plotly Documentation](https://plotly.com/python/pie-charts/)
* [MongoDB Documentation](https://www.mongodb.com/docs/manual/)
* [Dash Documentation](https://dash.plotly.com/)

**Steps to Complete Project**

**Installation**

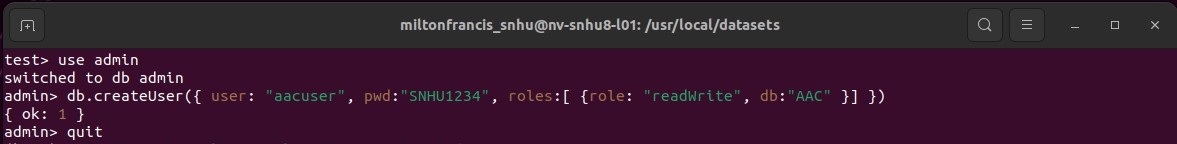
* Install MongoDB, at <https://www.mongodb.com>
* Install Python, at <https://www.python.org/downloads/>
* Download the Austin Animal Center data set in this repository
* Download the **‘aac\_crud.py’** Python script
* Import Austin Animal Center Outcomes data set:

A screenshot of a computer program

Description automatically generated

This screenshot shows changing into the directory where the data set is stored and will vary depending on where the file is saved. Importing the data set is accomplished with the ‘mongoimport’ command. The username, password, port, and host fields are held in constants that are applicable to my specific account. Naming the database is accomplished using ‘--db AAC' and naming the collection is accomplished using ‘--collection animals’. Finally, upon entering this command, the timestamp lines show the connection information and subsequent successful import of the documents.

* Create a new user account:



This screenshot shows changing databases to the ‘admin’ database and using ‘.createUser’ function to create a new user, named ‘aacuser’, with read and write permissions to the database AAC that was imported in the previous screenshot.

**Dashboard Development**

* Using Jupyter Notebook, update the connection variables to the specific variables for your MongoDB, seen here:

A computer screen with a black background

Description automatically generated

* The data table displays the information obtained from the data set. This can be modified to show a larger page size, or other options. This section is shown in the block below:

A screen shot of a computer code

Description automatically generated

* Each radio button corresponds to a set query that corresponds to the applicable filter-type. Initially, the value is set as ‘Reset’ to show all of the available dogs in the database.

A screen shot of a computer

Description automatically generated

* This interacts with a callback that has specific queries based on which button is selected. Each filter-type has a different pre-written query. This is shown here:

A screen shot of a computer program

Description automatically generated

* The pie chart also uses a callback to display the relevant information from the data set. This is modified in this section:

A screen shot of a computer code

Description automatically generated

* Finally, the map takes data from the selected row of the data table. This updates as the row selection changes and displays a pin of where the animal is located.

A computer screen with colorful text

Description automatically generated

**Testing**

* The dashboard was then tested to discover errors. Among the errors found, empty datasets proved to cause issues at times, so this was corrected with a proper check.
* Debug the application and ensure it functions as intended. Make special care to identify any edge cases for the requirements.
* Jupyter Notebook provided a simple place to test and iterate on my improvements.

**Challenges**

* I encountered multiple callback errors upon first running the application. The solution was adding check for ‘None’ values to ensure these errors were handled correctly.
* The pie chart took quite a bit of time to properly implement. This was due to the size of the database. Ideally, the database could be better sorted to remove redundancies. The solution in my application was to hide the text/legend space for the small slices of the pie chart. This allowed the chart to be at the correct size, as the legend egregiously overflowed otherwise.
* In the beginning of this project, establishing a stable connection with the MongoDB was tedious. This was improved by passing the connection variable into the CRUD module using Jupyter Notebook, rather than including the variables hard-coded in the module.

**Contact**

**Milton Francisco**