**README: CS-340 Dashboard Project**

**Project Overview**

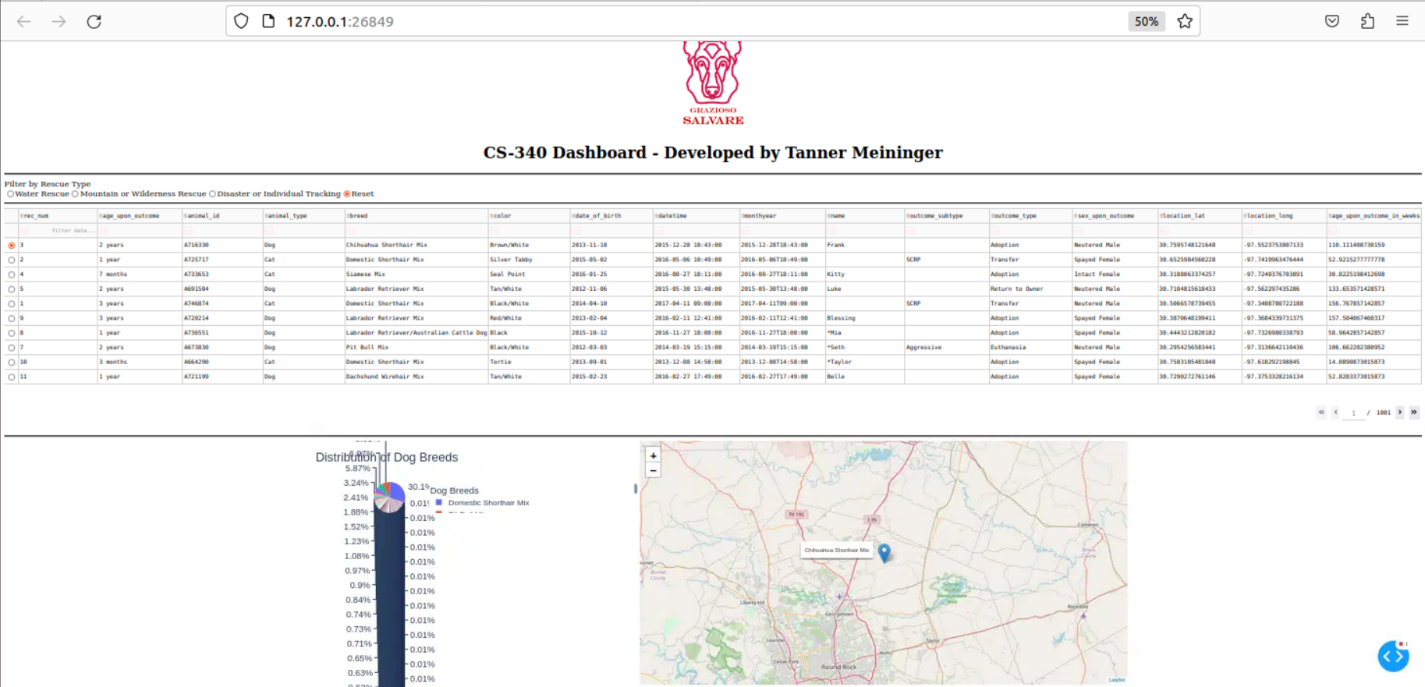
This project involves creating an interactive dashboard for Grazioso Salvare that visualizes rescue operation data for dogs. The dashboard allows the client to filter data dynamically, visualize it in a pie chart and a geolocation map, and interact with it through a responsive data table. The data is retrieved from a MongoDB database, and the web application is built using the Dash framework.

**Key Functionalities**

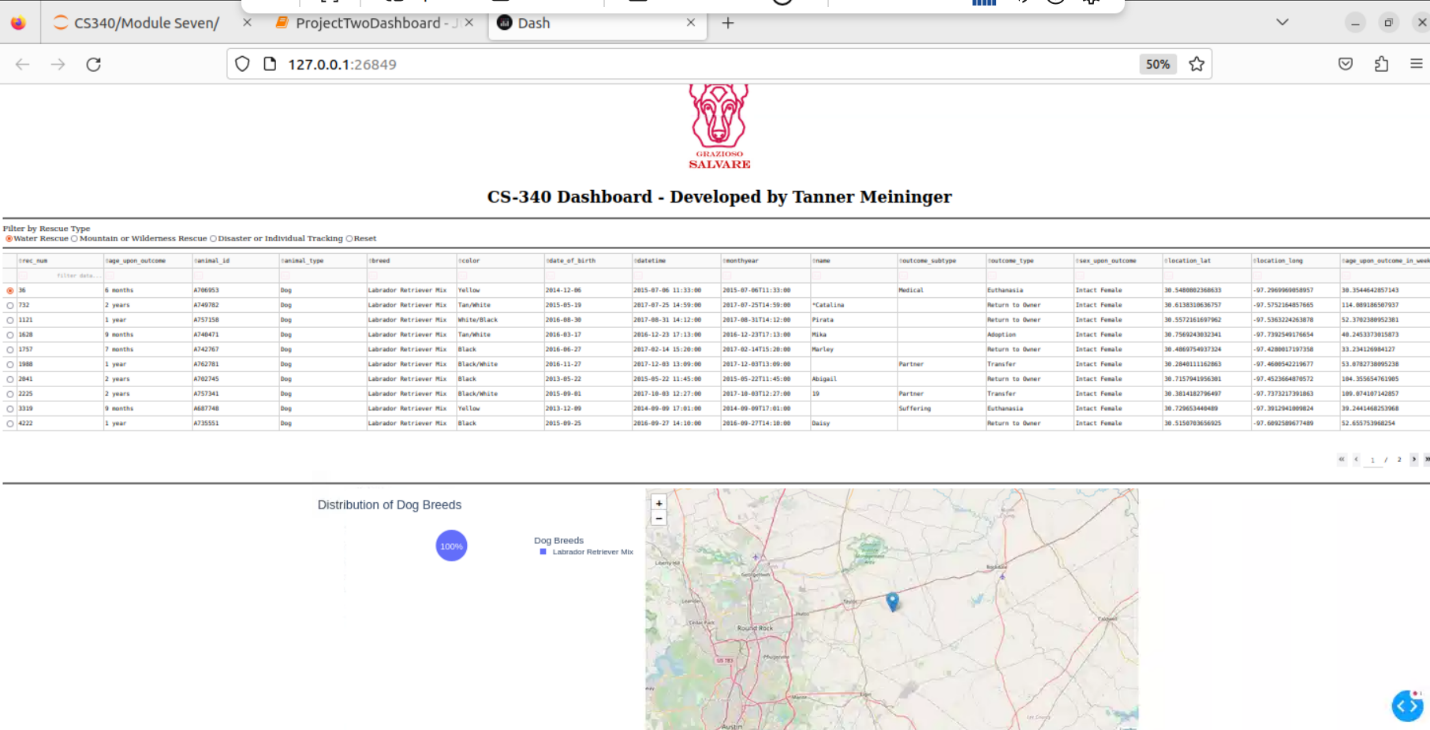
* **Interactive Filters**: Users can filter the data by rescue type (Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking), and reset the filters to view all data.
* **Data Table**: A dynamic data table that updates based on the selected filter and displays the relevant records.
* **Pie Chart**: A pie chart displaying the distribution of dog breeds, dynamically updated based on the filter applied.
* **Geolocation Map**: A map displaying the geographical location of selected dogs based on the data table.

**Screenshots:**

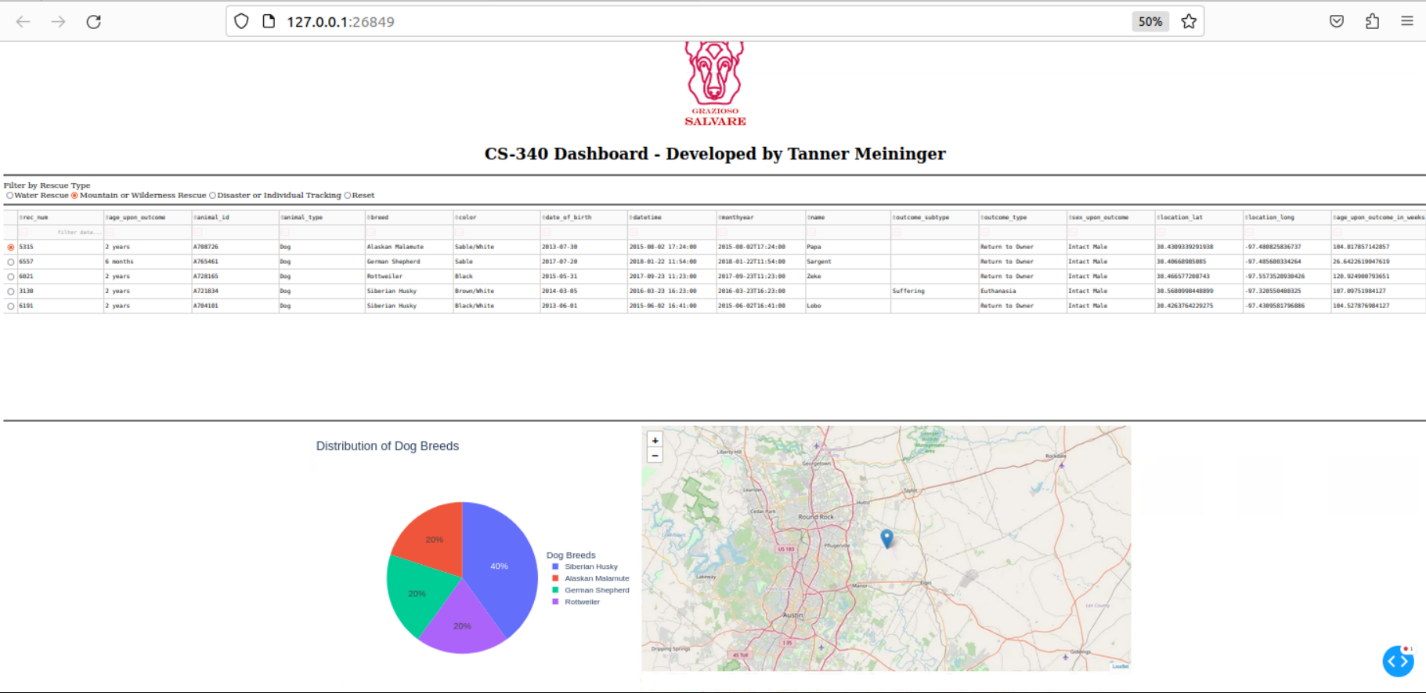
* Initial state of the dashboard with **no filters (and reset filter)** applied.



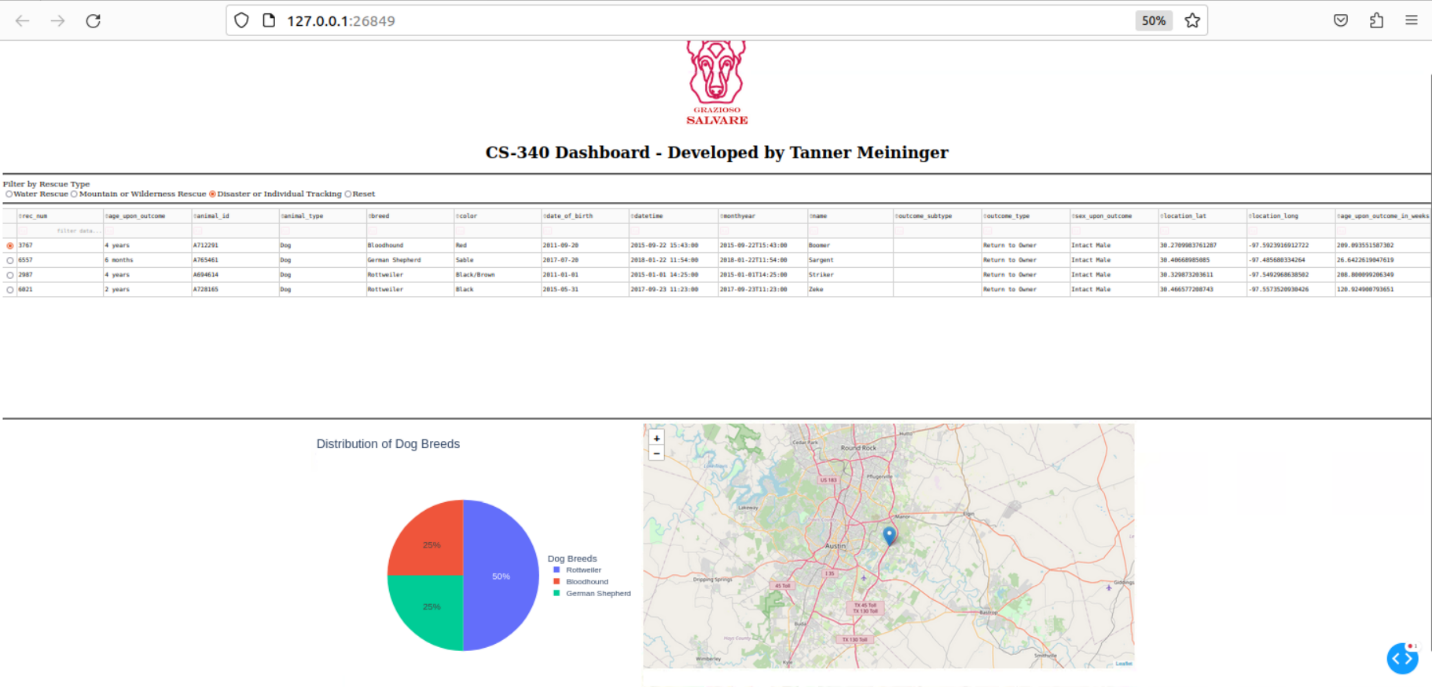
* Dashboard after applying each of the following filters:
  + **Water Rescue**



* + **Mountain or Wilderness Rescue**



* + **Disaster or Individual Tracking**



**Tools and Technologies Used**

**MongoDB**

* **Purpose**: MongoDB was used as the model component of the project to store and manage the rescue operation data.
* **Why MongoDB?:**
  + Scalability: MongoDB is a NoSQL database that excels at handling large datasets and provides flexibility in data structure.
  + Ease of Use with Python: MongoDB provides an excellent interface for Python using the pymongo library, which allows for easy CRUD (Create, Read, Update, Delete) operations.
  + Document-Oriented: Its document-oriented nature is ideal for storing semi-structured data like rescue records that may include complex relationships or varying attributes.

**Dash Framework**

* **Purpose**: Dash is a Python framework used for building web applications with an interactive user interface (UI). It provides both the view and controller structure for the project.
* **Why Dash?**
  + Python Integration: Dash allows seamless integration with Python, making it ideal for data-driven applications where Python is used for data manipulation and visualization.
  + Interactive Components: Dash provides various UI components such as graphs, data tables, and filters that automatically update when new data is passed to them.
  + Plotly for Visualizations: Dash integrates with Plotly to create dynamic and interactive charts, which was essential for the pie chart and map used in this project.

**Other Tools:**

* **Dash Leaflet**: Used for the geolocation map visualization.
* **Plotly**: Used for creating the interactive pie chart that visualizes dog breeds.
* **JupyterDash**: A version of Dash that allows for running Dash applications within Jupyter Notebooks.
* **Jupyter Notebooks**

**Steps to Reproduce the Project**

**Set Up MongoDB**

* Install MongoDB and load the rescue data into the database.
* Configure a MongoDB user account (e.g., aacuser) with the necessary permissions.
* Ensure the MongoDB instance is running and accessible from your application.

**Create the CRUD Python Module**

* Develop the CRUD class to handle interactions with MongoDB (insert, read, update, and delete operations).
* Test the CRUD operations to ensure data is correctly retrieved from the database.

**Develop the Dashboard Application**

* Use the Dash framework to build the user interface, including the data table, pie chart, and map.
* Use Plotly for creating a dynamic pie chart visualizing dog breed distribution.
* Use Dash Leaflet to render the geolocation map that shows where each dog was rescued.
* Set up filters (radio buttons) to allow users to filter data based on rescue type.

**Run the Application**

* Use JupyterDash to run the Dash application in a Jupyter Notebook or as a standalone web app.
* Test the application to ensure the filters, charts, and maps are responsive and update dynamically based on user input.

**Test and Capture Screenshots**

* Ensure all functionality is working as expected (filters, chart updates, data table, map).
* Capture screenshots or screencast for the project documentation.

**Challenges and Solutions**

**Challenge 1: MongoDB Authentication and Connection**

* **Issue**: There were initial difficulties in connecting to MongoDB due to authentication issues.
* **Solution**: The problem was resolved by ensuring that the correct MongoDB URI was used, with proper encoding for special characters in the username and password.

**Challenge 2: Plotly Legend Overlapping the Chart**

* **Issue**: The legend in the pie chart was overlapping the chart itself, making it difficult to read.
* **Solution**: The legend was moved to the right of the pie chart and resized for better readability.

**Challenge 3: Data Filtering**

* **Issue**: Ensuring the correct filtering logic was applied based on rescue type.
* **Solution**: Applied MongoDB queries that filter the data by breed, age, and sex to match the rescue type criteria.