Austin Antles

CS 340

June 19 2025

**Dashboard README**

## Project Functionality

The purpose of this project is to provide Grazioso Salvare with an interactive web dashboard to help identify suitable dogs from the Austin Animal Center dataset for search-and-rescue training. The application allows users to filter and visualize data based on criteria relevant to different types of rescue operations.

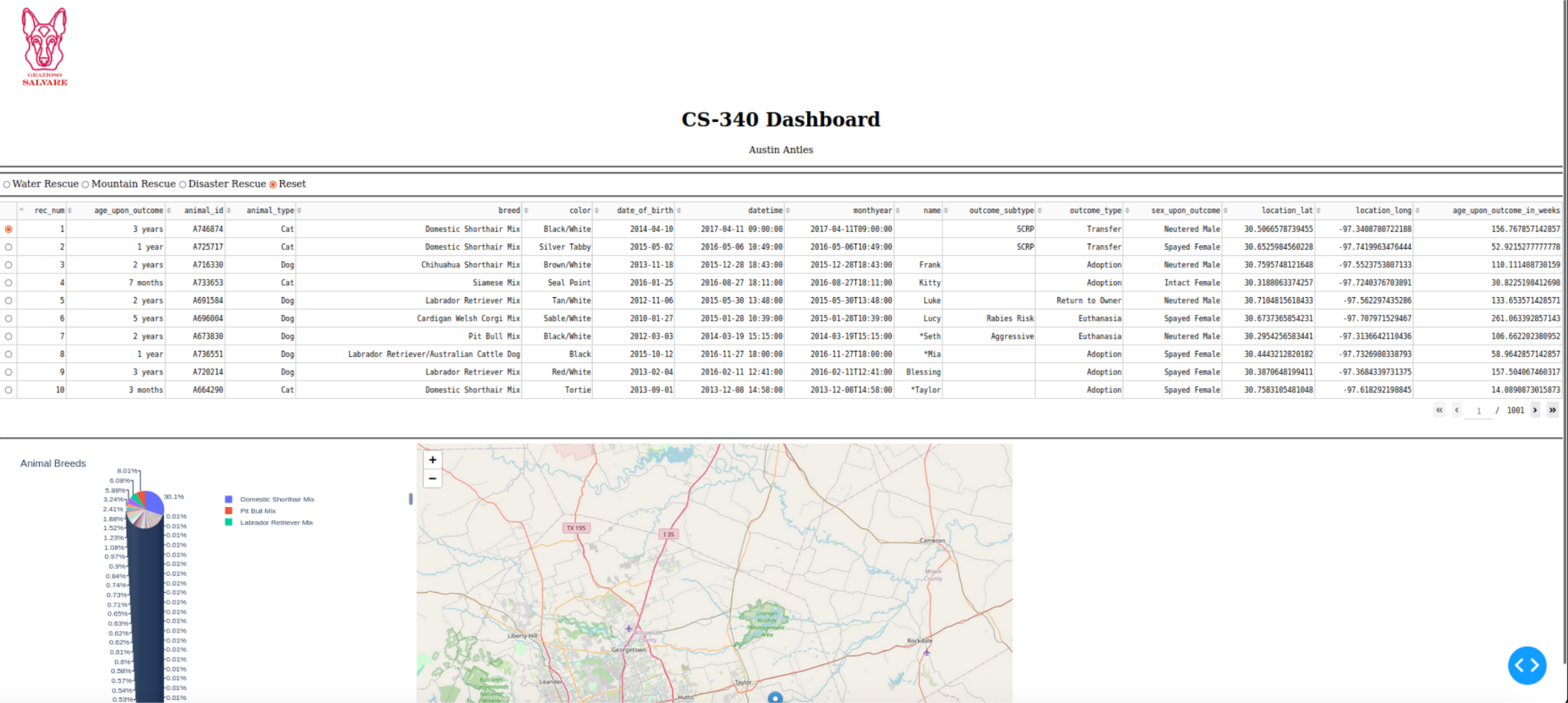
The core functionalities of the dashboard are:

1. **Branding and Identification**: The dashboard prominently displays the Grazioso Salvare logo, which links to the client's homepage, and includes the developer's name as a unique identifier.
2. **Interactive Filtering**: Users can filter the animal dataset using radio buttons for specific rescue categories:
   * Water Rescue
   * Mountain or Wilderness Rescue
   * Disaster or Individual Tracking
   * A "Reset" option to view the complete, unfiltered dataset.
3. **Dynamic Data Table**: A table displays animal data that dynamically updates in response to the selected filter. This table is user-friendly, featuring pagination, multi-column sorting, and single-row selection.
4. **Dynamic Charts**:
   * **Geolocation Map**: An interactive map that displays the location of a selected animal from the data table.
   * **Breed Distribution Pie Chart**: A pie chart that shows the distribution of animal breeds within the currently filtered dataset, updating dynamically with the table.

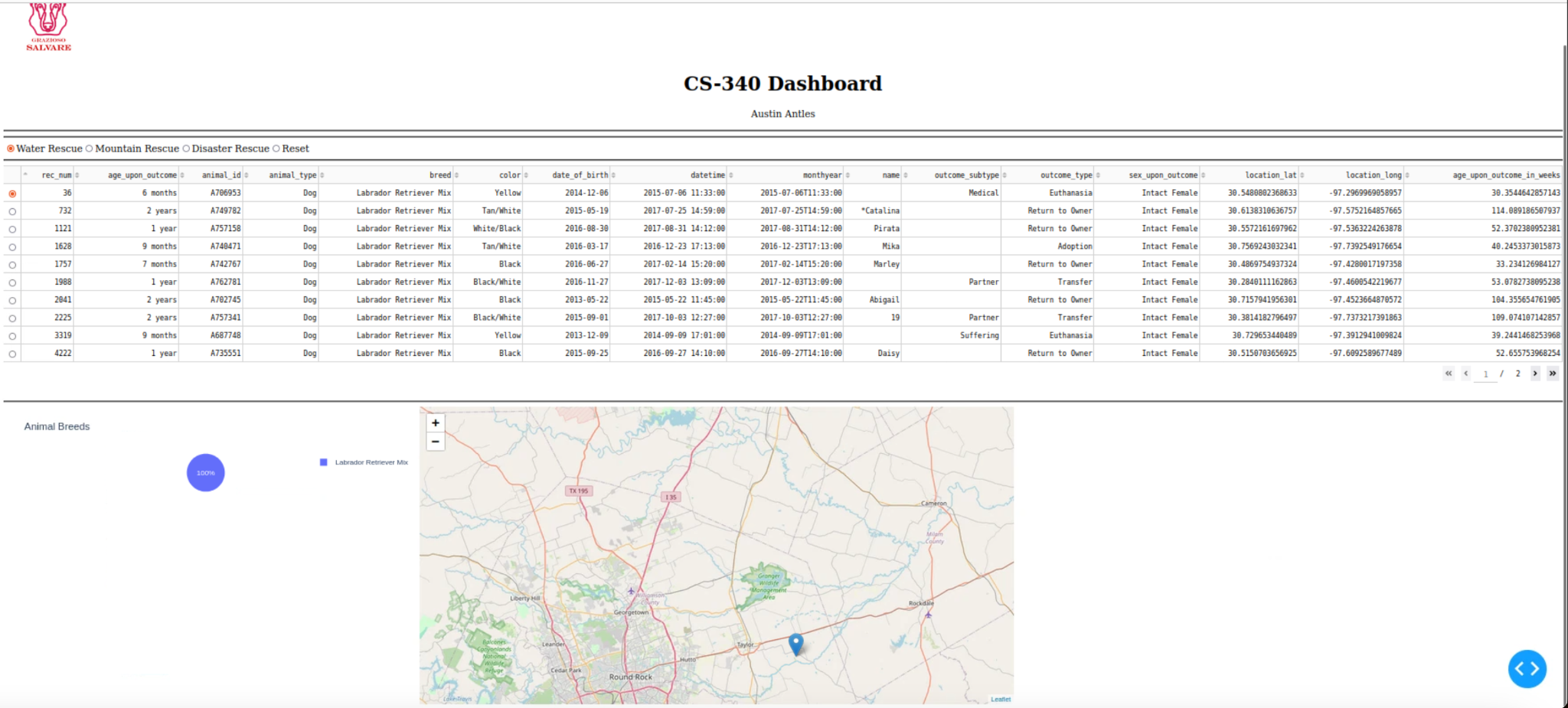
### Dashboard Functionality Showcase

Below are screenshots demonstrating the dashboard in its various states.

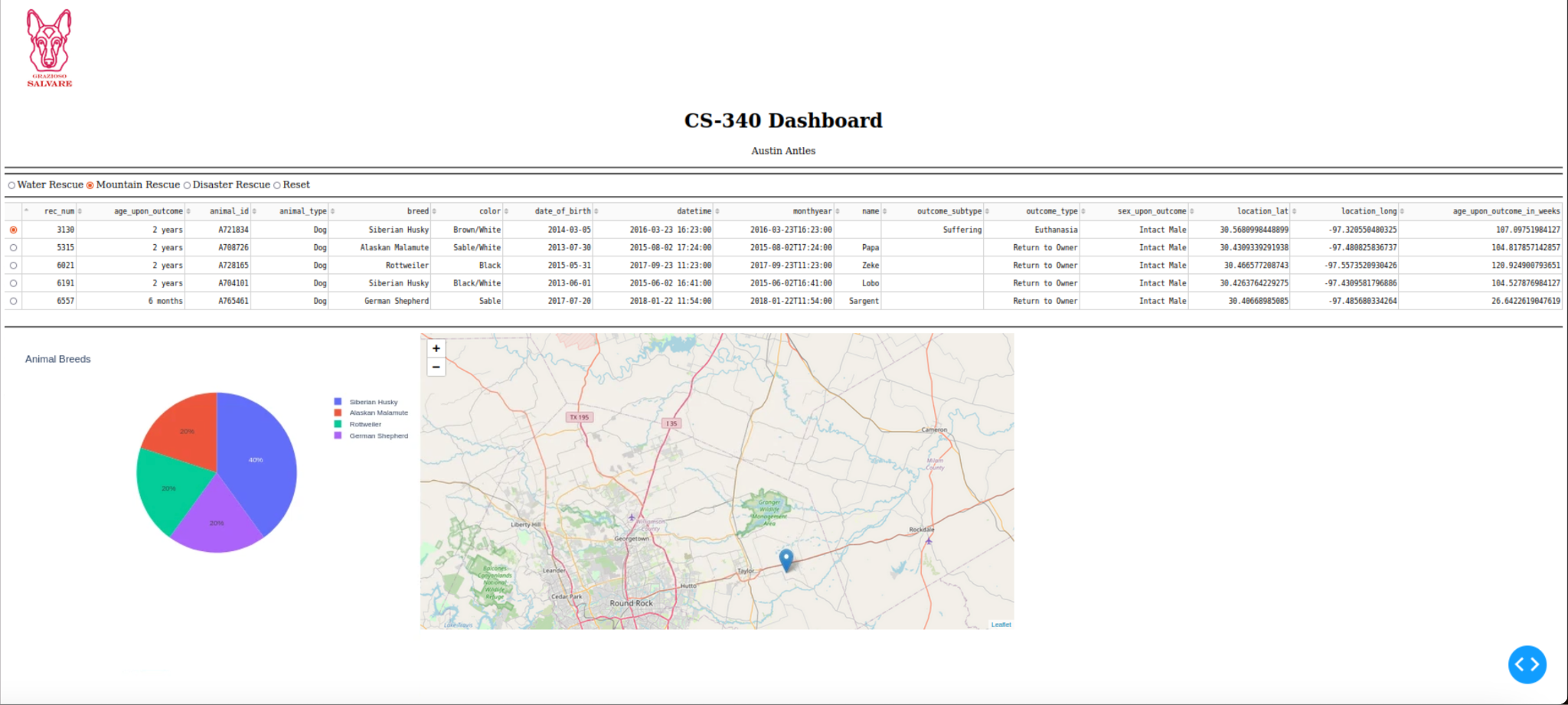
**1. Initial Unfiltered State:** *(This view shows all data loaded into the interactive table, with the pie chart and map reflecting the default state.)*

**

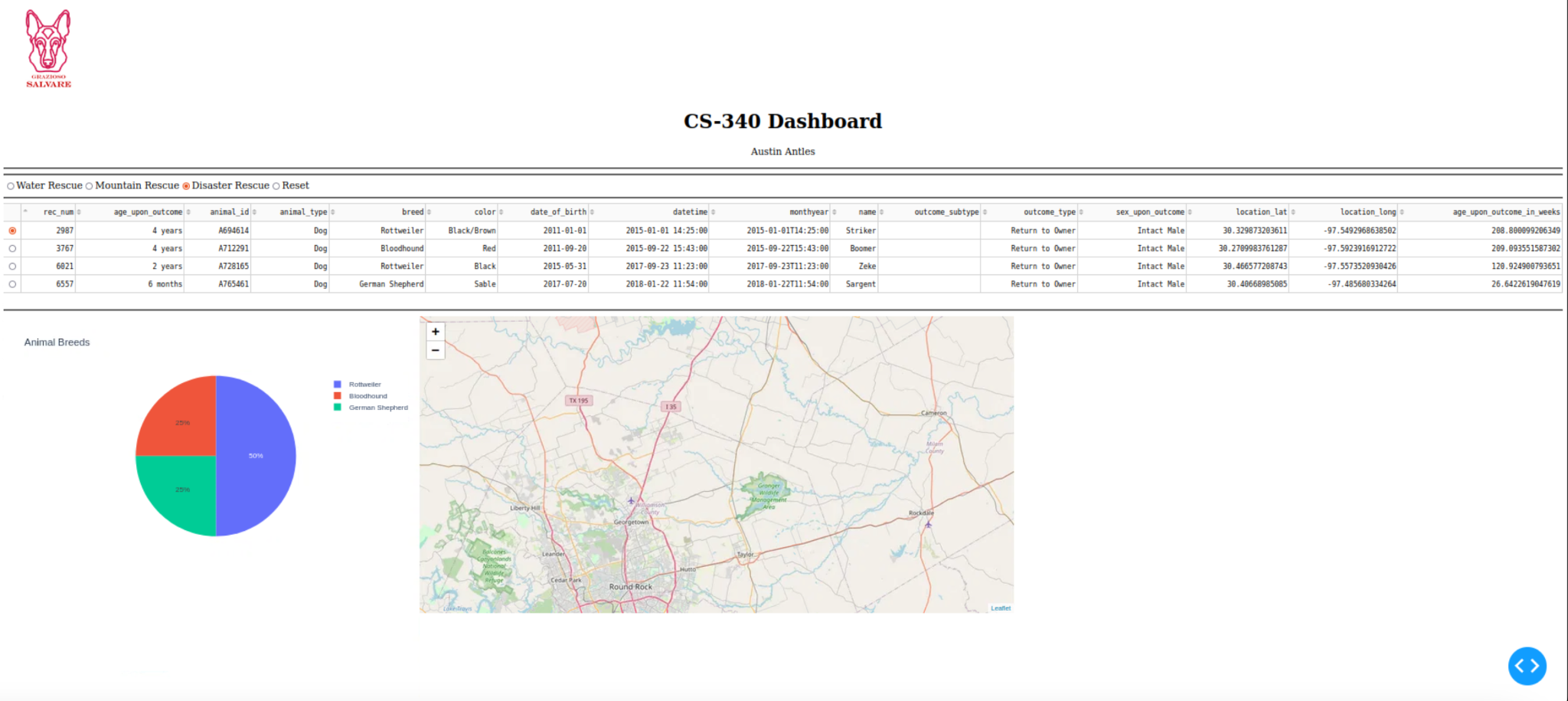
**2. "Water Rescue" Filter Applied:** *(This view shows the data table and charts filtered to display only dogs suitable for water rescue operations.)*

**

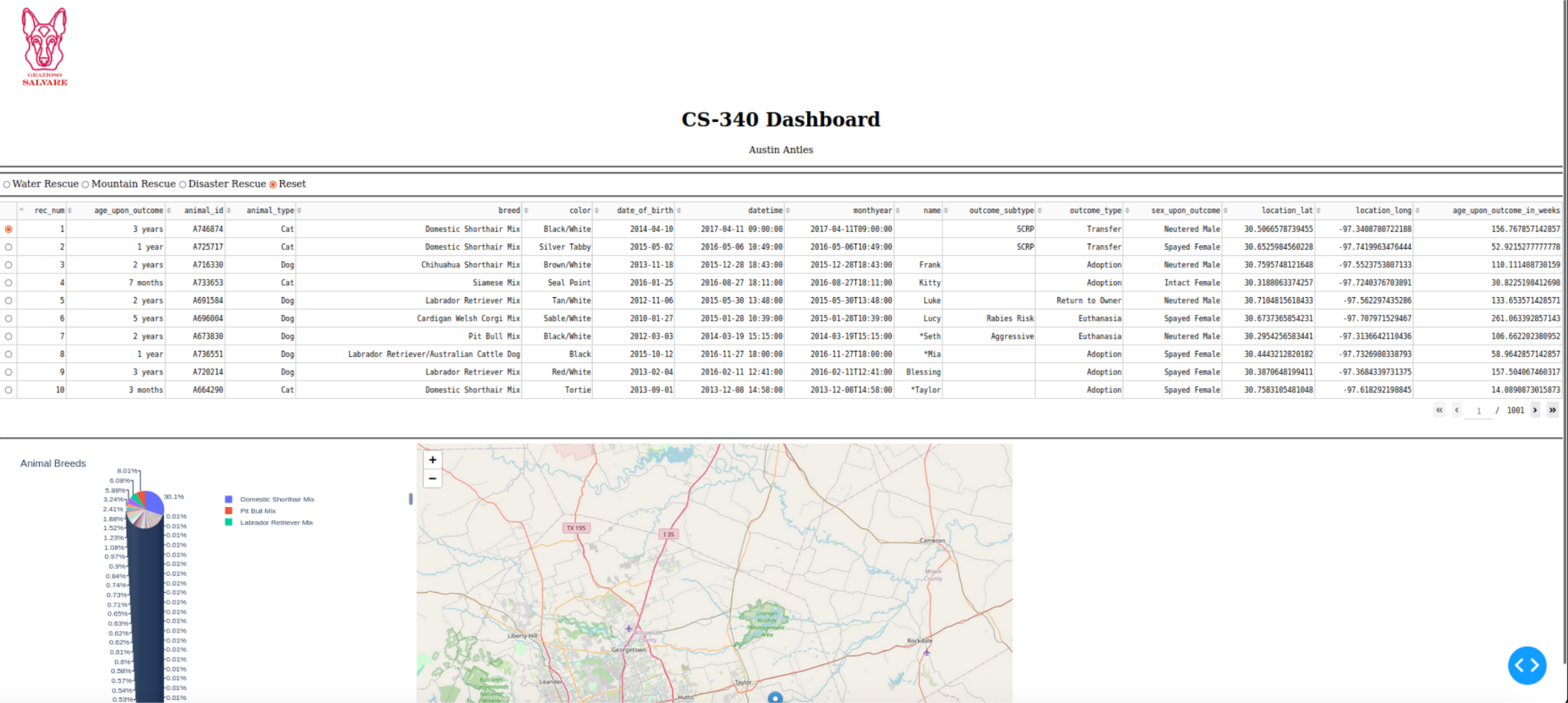
**3. "Mountain or Wilderness Rescue" Filter Applied:** *(This view shows the data table and charts filtered for dogs suitable for mountain or wilderness rescue.)*

**

**4. "Disaster or Individual Tracking" Filter Applied:** *(This view shows the data table and charts filtered for dogs suitable for disaster response or tracking.)*

**

**5. "Reset" Applied:** *(This view demonstrates the dashboard after the "Reset" option is selected, returning all widgets to their original, unfiltered state.)*

**

## Tools, Technologies, and Rationale

The project was developed using a Model-View-Controller (MVC) architectural pattern, implemented with the following tools:

* **Model (Database): MongoDB**
  + **Description**: MongoDB is a source-available, cross-platform, document-oriented NoSQL database program. It uses JSON-like documents with optional schemas (which it calls BSON).
  + **Rationale**: MongoDB was chosen for its flexibility and seamless integration with Python. Unlike traditional SQL databases, its schema-less nature is highly advantageous for handling complex, varied datasets, such as animal shelter records, which may have inconsistent or evolving fields. The pymongo library for Python provides a very natural, dictionary-like interface for performing CRUD (Create, Read, Update, Delete) operations, making it straightforward to pass data between the database and the Python backend without extensive object-relational mapping.
* **View & Controller (Web Framework): Plotly Dash**
  + **Description**: Dash is an open-source Python framework for building analytical web applications. It is built on top of Flask, Plotly.js, and React.js.
  + **Rationale**: Dash was the ideal choice because it allows for the creation of complex, interactive dashboards using pure Python. This eliminates the need for writing separate JavaScript for front-end interactivity.
    - The **View** (layout) is constructed using Python components from dash\_html\_components and dash\_core\_components, which translate directly to HTML and React components in the browser.
    - The **Controller** (interactivity) is managed through Dash's callback functions. These Python functions are triggered by user interactions (like clicking a filter button or selecting a row in a table). They can update any property of any component in the layout. This declarative style makes the application logic easy to understand and maintain.
* **Data Manipulation: Pandas**
  + **Description**: Pandas is a powerful Python library for data manipulation and analysis.
  + **Rationale**: While pymongo retrieves data from the database, Pandas is used to structure this data into DataFrames. This is the ideal format for use with Dash's DataTable and for creating visualizations with Plotly Express, simplifying tasks like data cleaning, transformation, and aggregation.

### Software and Resources Used

* **Python 3**: The core programming language for the entire project. [python.org](https://www.python.org/)
* **MongoDB**: The NoSQL database used to store and query the animal shelter data. [mongodb.com](https://www.mongodb.com/)
* **Plotly Dash**: The framework used to build the interactive web dashboard. [dash.plotly.com](https://dash.plotly.com/)
* **Pandas**: The library used for data manipulation and analysis. [pandas.pydata.org](https://pandas.pydata.org/)
* **Jupyter Notebook**: The development environment used to create and run the dashboard application interactively. [jupyter.org](https://jupyter.org/)

## Project Completion Steps

1. **Environment Setup**: A local Python environment was configured with all necessary libraries (jupyter-dash, pandas, pymongo). A MongoDB server was set up, and the Austin Animal Center Outcomes dataset (aac\_shelter\_outcomes.csv) was imported into a collection.
2. **CRUD Module Creation**: A Python module (animal\_shelter.py) was developed. This module contains the AnimalShelter class, which encapsulates all the necessary CRUD (Create, Read, Update, Delete) operations for connecting to and interacting with the MongoDB database. This modular approach separates database logic from the application logic.
3. **Dashboard Scaffolding**: The initial layout of the Dash application was created in a Jupyter Notebook. This included creating an unfiltered, paginated, and sortable DataTable to display all animal records.
4. **Implementing Filter Queries**: The specific database queries required for the Water, Mountain, and Disaster rescue filters were developed and tested within the AnimalShelter module and then integrated into the dashboard.
5. **Building Interactivity with Callbacks**:
   * A callback function was created to connect the radio button filters to the DataTable. This function re-queries the database based on the selected filter and updates the table's data.
   * Another callback was added to generate the pie chart. It uses the data currently visible in the DataTable (derived\_virtual\_data) as its input, ensuring it always reflects the user's current view.
   * The map callback was implemented to update the map's center and marker based on the row selected by the user in the DataTable.
6. **Final Touches and Testing**: The Grazioso Salvare logo and unique developer identifier were added to the layout. The entire dashboard was thoroughly tested for functionality, ensuring that all filters functioned as expected and all components were updated correctly.

## Challenges and Solutions

* **Challenge: Database Connectivity**
  + **Description**: Initially, ensuring a stable connection between the Python script and the MongoDB server required careful configuration of the host, port, username, and password within the AnimalShelter class constructor.
  + **Solution**: The connection parameters were abstracted into variables, making them easy to modify and update. The AnimalShelter class was designed to handle connection details, allowing the main dashboard application code to remain clean and focused on its primary logic.
* **Challenge: Data Serialization for Dash Callbacks**
  + **Description**: A significant technical challenge arose when filtering the data. The application would crash with a TypeError: Object of type ObjectId is not JSON serializable. This occurred because the data returned from MongoDB includes an \_id field, which is a special BSON ObjectId type. Dash callbacks can only handle standard JSON-serializable data types (strings, numbers, lists, dictionaries).
  + **Solution**: The fix was to ensure that the \_id column was explicitly removed from the data *before* it was returned from the callback function to the DataTable. This was achieved by converting the query result to a Pandas DataFrame and using the df.drop(columns=['\_id'], inplace=True) method. This step ensures that the data passed to the front end is clean and serializable.