**Grazioso Salvare Animal Rescue Dashboard**

Project Author: Nurlan Gasimzade

Course: CS 340 - Client/Server Development

**Project Functionality**

This project is an interactive web dashboard created for the international rescue-animal training company, Grazioso Salvare. The application provides a user-friendly interface to access and visualize data from five animal shelters in the Austin, Texas, region.

The primary function of this dashboard is to allow users at Grazioso Salvare to filter a large dataset of shelter animals to quickly identify dogs that meet specific criteria for different types of search-and-rescue training. The dashboard allows filtering by Water Rescue, Mountain/Wilderness Rescue, and Disaster/Individual Tracking profiles, each with its own unique requirements for breed, age, and sex.

The application displays the filtered data in an interactive table and two dynamic charts: a pie chart showing the distribution of breeds and a geolocation map showing the location of a selected animal.

**Proof of Functionality**

The following screenshots demonstrate the dashboard in its various functional states.

Figure 1: Initial "Reset" View

(This view displays all unfiltered data from the shelter database.)

**A screenshot of a computer

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: "Water Rescue" Filter Applied

(This view displays only dogs that meet the Water Rescue criteria: specific breeds, intact females, and within the required age range.)

**A screenshot of a computer

AI-generated content may be incorrect.**A blue circle with white text

AI-generated content may be incorrect.

Figure 3: "Mountain or Wilderness Rescue" Filter Applied

(This view displays only dogs that meet the Mountain/Wilderness Rescue criteria.)

**A screenshot of a computer

AI-generated content may be incorrect.**

A pie chart with different colored circles

AI-generated content may be incorrect.

Figure 4: "Disaster or Individual Tracking" Filter Applied

(This view displays only dogs that meet the Disaster/Individual Tracking criteria.)

**A screenshot of a computer

AI-generated content may be incorrect.**

A pie chart with different colors

AI-generated content may be incorrect.

**Tools, Technologies, and Rationale**

The application was developed using an MVC (Model-View-Controller) design pattern. The following tools and technologies were chosen to build a robust and efficient client/server application.

**Model Component: MongoDB**

The database, which serves as the "Model," was implemented using **MongoDB**. This NoSQL, document-based database was chosen for several key reasons:

* **Flexibility:** MongoDB's schema-less nature is ideal for handling real-world data from animal shelters, which can often be inconsistent or have evolving fields.
* **Performance:** It is highly efficient at handling large volumes of data and complex queries.
* **Python Integration:** It integrates seamlessly with Python through the **PyMongo** driver, which allowed for the creation of a clean and object-oriented CRUD Python module (animal\_shelter.py) to handle all database interactions.

**View and Controller: Python with Dash**

The dashboard itself, which represents the "View" and "Controller," was built entirely in Python using the **Dash by Plotly** framework.

* **Python:** The core language for its extensive data science libraries and clear, readable syntax. The **Pandas** library was used to transform data retrieved from MongoDB into structured DataFrames, which are the required format for the dashboard's table and charts.
* **Dash Framework:** This framework was chosen because it allows for the creation of powerful, interactive web applications using only Python, eliminating the need to write separate HTML, CSS, or JavaScript.
  + **View:** The layout was constructed using components from dash.html (for structure like headers and divs) and dash.dcc (for interactive elements like radio buttons and graphs).
  + **Controller:** The application's interactivity is powered by @app.callback functions. These functions "listen" for user input (like clicking a filter button) and trigger Python code to query the database and update the dashboard components in real-time.

**Instructions for Reproduction**

To run this project, the following steps must be taken:

1. **Environment:** The project is designed to run in the SNHU Apporto virtual lab environment, which has Python and the necessary libraries pre-installed.
2. **Files:** Place the following files into a single directory:
   * ProjectTwoDashboard (1).ipynb (the main notebook file)
   * animal\_shelter.py (the CRUD module)
   * mongodb-env.txt (containing the database credentials)
   * A folder named assets containing Grazioso Salvare Logo.png
3. **Dependencies:** Ensure the following Python libraries are installed: jupyter\_dash, pandas, pymongo, dash, plotly.
4. **Execution:** Open and run the ProjectTwoDashboard (1).ipynb notebook. The application will connect to the remote database and display the interactive dashboard directly below the code cell.

**Challenges Encountered and Solutions**

Several significant technical challenges were encountered during the development of this project.

1. **Challenge: Database Connection Errors**
   * **Problem:** Initially, the application could not connect to the database, producing a series of errors from Connection refused to AuthenticationFailed. Debugging revealed that the database was not running locally, but was a remote server.
   * **Solution:** After extensive troubleshooting, a credentials file (mongodb-env.txt) was located. The connection string in the animal\_shelter.py module was updated to use the correct remote Host, Port, Username, and Password. A final authentication issue was resolved by specifying the correct authSource ('admin') in the MongoDB connection URI, allowing the root user to authenticate properly before accessing the AAC database.
2. **Challenge: Application Stability**
   * **Problem:** During development, the web server would often fail to shut down correctly, leading to an Address already in use error upon restarting the application.
   * **Solution:** This was resolved by modifying the final line of the notebook to launch the server on a new, random port for each session (app.run\_server(mode='inline', port=random.randint(8000, 8999))). This bypasses any issues with stuck processes occupying a specific port.
3. **Challenge: Handling Incomplete Data**
   * **Problem:** Some animal records in the database were missing GPS location data. This had the potential to crash the application when trying to display the map.
   * **Solution:** The map's callback function was written to be "data-aware." It first checks if the selected animal record contains valid latitude and longitude data. If the data is missing, instead of crashing, it gracefully displays a message ("Location data not available for this entry") on a blank map, ensuring a robust and user-friendly experience.