Grazioso Salvare – Animal Shelter Dashboard

This interactive dashboard was developed for Grazioso Salvare, a rescue-focused animal shelter that places dogs into specialized operations such as water rescue, wilderness tracking, and disaster recovery. The dashboard allows staff to filter animals by rescue suitability, view breed statistics, and see animal location data through an interactive web interface.

# Required Functionality

The dashboard fulfills all functional requirements:  
  
- Dropdown Filter: Users can select from Water Rescue, Wilderness Rescue, Disaster Rescue, or All Types to filter the data.  
- Data Table: The table updates automatically with the filtered results and includes sorting, filtering, pagination, and cell highlighting.  
- Pie Chart: Breed distribution is visualized in an interactive chart. When All Types is selected, the chart displays only the top 10 breeds.  
- Geolocation Map: The selected animal’s latitude and longitude are plotted on a Leaflet map.  
- Responsive Styling: All UI elements are aligned for readability and accessibility in Jupyter.

**Screenshots:**

**Initial State:**

**A screenshot of a computer

Description automatically generated**

**Water Rescue Filter:**

**A screenshot of a computer

Description automatically generated**

**Wilderness Rescue Filter:**

**A map of a road

Description automatically generated**

**Disaster Rescue Filter:**

**A screenshot of a computer

Description automatically generated**

# Tools and Technologies Used

- Dash: Built interactive UI using Python — handles both the View and Controller logic.  
- Dash Leaflet: Provided geolocation mapping using Leaflet.js inside Dash.  
- MongoDB: Used as the NoSQL document database to store animal records.  
- PyMongo: Allowed for direct MongoDB queries from within Python.  
- Pandas: Processed and cleaned MongoDB query results.  
- Plotly Express: Created responsive and interactive pie charts.  
- JupyterDash: Allowed the Dash app to run directly in Jupyter Notebooks for easier testing.

# Why MongoDB Was Used

MongoDB was chosen as the Model layer of the MVC pattern for several reasons:  
  
- It stores animal records as JSON-like documents, which are naturally compatible with Python dicts.  
- Its query language allows powerful and readable filters using operators like $in, $gte, $lte, and $set.  
- It scales well and supports flexible schemas — ideal for evolving animal data fields.  
- It interfaces directly with Python through PyMongo, making CRUD operations seamless.

# Dash Framework as View + Controller

The Dash framework serves both the View and Controller in this MVC-based architecture:  
  
- The View is created using Dash components like html.Div, dcc.Dropdown, dash\_table.DataTable, and dcc.Graph.  
- The Controller logic is implemented using @app.callback functions that connect user input to backend logic and update the display.

# Project Steps

1. Created a MongoDB CRUD class AacCrud.py to abstract the database operations.  
2. Built the dashboard layout with Dash components.  
3. Connected the DataTable, Dropdown, Pie Chart, and Map using callback functions.  
4. Filtered data based on dropdown selection and updated visual elements accordingly.  
5. Validated MongoDB queries and handled edge cases (e.g. empty selection, missing coordinates).  
6. Styled layout for readability and included a clickable logo.  
7. Tested all functionality in JupyterDash.

# CRUD Module Overview

The AacCrud.py module includes four key methods for database interaction:  
  
db.create(data) # Inserts a new animal record  
db.read(queryParams) # Retrieves records matching filters  
db.update(query, updates) # Updates matching records with new values  
db.delete(query) # Deletes all records matching the query

# Challenges & Solutions

- Query filters returned no results → Validated field names and values inside MongoDB; confirmed sex and breed values were case-sensitive.  
- Dropdown stretched across layout → Limited width using inline CSS: 'width': '200px'.  
- Map error on unselected row → Handled None or empty index list gracefully with default fallback row.  
- MongoDB \_id field crash → Dropped \_id field before loading into DataTable using df.drop().  
- Inconsistent column indices in map logic → Verified and hardcoded column indices based on field positions in returned data.

# Resources Used

- https://dash.plotly.com/  
- https://github.com/python-visualization/dash-leaflet  
- https://pymongo.readthedocs.io/en/stable/  
- https://www.mongodb.com/docs/manual/aggregation/  
- https://pandas.pydata.org/