

Grazioso Salvare Rescue Dog Dashboard

README & Project Documentation

Jeremy Ritchie | CS-340 Client/Server Development | SNHU

Author	Jeremy Ritchie
Course	CS-340: Client/Server Development
Institution	Southern New Hampshire University (SNHU)
Client	Grazioso Salvare (via Global Rain)
Stack	Python, Dash, MongoDB, Plotly, Dash Leaflet
Database	MongoDB 7.0 — aac.animals collection (20,000 records)

1. Project Overview

Grazioso Salvare is an international search-and-rescue training company that identifies dogs suitable for specialized rescue training. This project delivers a full-stack web dashboard that connects to a MongoDB database of Austin Animal Center shelter outcomes and allows Grazioso Salvare staff to interactively filter, explore, and visualize dog candidates by rescue type.

The application was built using the MVC (Model-View-Controller) architectural pattern:

- Model: MongoDB database accessed via a custom Python CRUD module (AnimalShelter class)
- View: Dash web framework rendering an interactive DataTable, pie chart, and geolocation map
- Controller: Dash callback functions linking user filter selections to data updates across all widgets

2. Required Functionality

2.1 Dashboard Branding

- Grazioso Salvare logo displayed at the top, wrapped in a hyperlink to www.snhu.edu
- Unique developer identifier: "Developed by Jeremy Ritchie | CS-340 | SNHU"

2.2 Interactive Filter Options

Radio button controls allow filtering of all dashboard widgets by the four required rescue types:

- Water Rescue — Labrador Retriever Mix, Chesapeake Bay Retriever, Newfoundland; Intact Female; 26–156 weeks
- Mountain or Wilderness Rescue — German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, Rottweiler; Intact Male; 26–156 weeks
- Disaster or Individual Tracking — Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler; Intact Male; 20–300 weeks
- Reset — returns all widgets to their original unfiltered state showing all 20,000 records

2.3 Interactive Data Table

- Displays Austin Animal Center Outcomes data loaded from MongoDB
- Dynamically updates when a rescue type filter is selected
- Features: pagination (10 rows/page), native column sorting, native column filtering, single-row selection, column highlight on click

2.4 Charts

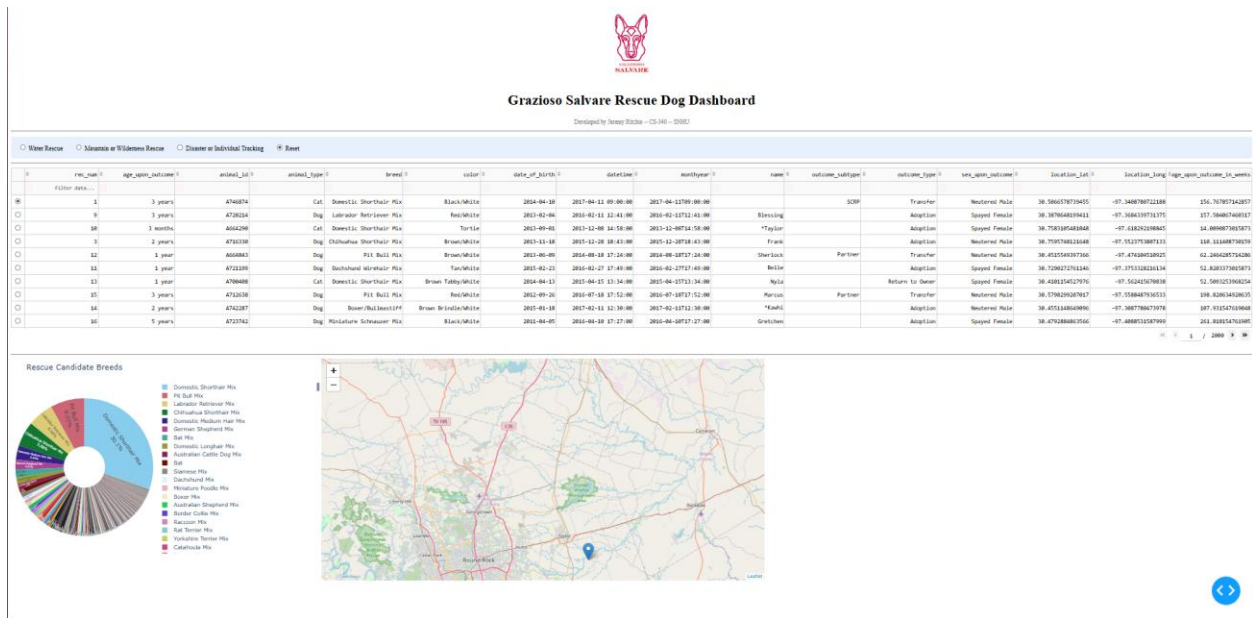
- Pie chart: displays breed distribution of the currently filtered dataset; updates dynamically with filters
- Geolocation map: renders an interactive Leaflet map centered on the selected row's GPS coordinates; shows animal name and breed in a popup marker

3. Dashboard Screenshots

Note: The screenshots below were taken during testing and deployment in the Codio/JupyterLab environment. Each screenshot includes the Grazioso Salvare logo and the developer unique identifier as required by the specification.

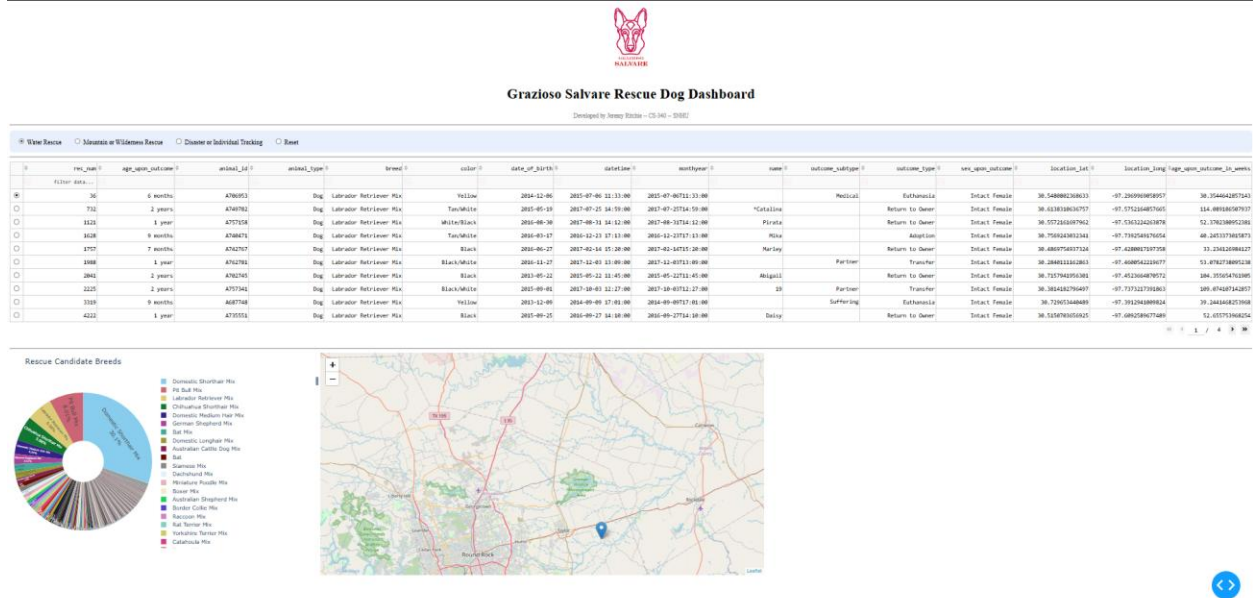
Screenshot 1 — Starting State (Reset / Unfiltered)

Shows the full unfiltered dataset (20,000 records across 2,000 pages). The pie chart displays all breeds and the map is centered on Austin, TX.



Screenshot 2 — Water Rescue Filter Applied

Filter returns 34 records matching Labrador Retriever Mix, Chesapeake Bay Retriever, and Newfoundland breeds; Intact Female; age 26–156 weeks. Pie chart and map update accordingly.



Screenshot 3 — Mountain or Wilderness Rescue Filter Applied

Filter returns records matching German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, and Rottweiler breeds; Intact Male; age 26–156 weeks.

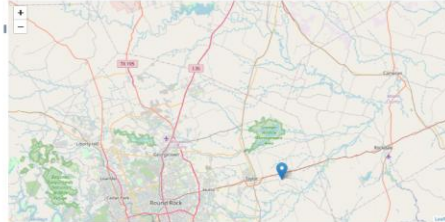
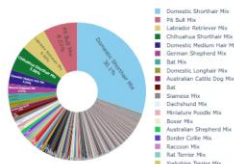


Grazioso Salvere Rescue Dog Dashboard

Developed by Nancy Ritchie - CS 460 - 55552

View Rescue <input checked="" type="radio"/> <input type="radio"/> Disaster or Wilderness Rescue <input type="radio"/> Disaster or Individual Tracking <input type="radio"/> Basic													
id	rescue_name	age_weeks_outcome	animal_id	animal_type	breed	color	date_of_birth	date_lost	month/year	name	outcome_subtype	outcome_type	sex_age_outcome
3138	2 years	AT2834	Dog	Silver Lab Husky	Brown/White	2004-03-05	2004-03-23 04:23:00	2004-03-23 04:23:00			Suffering	Euthanasia	Intact Male
5105	2 years	AT0702	Dog	Alaskan Malamute	Sable/White	2003-07-20	2003-09-02 07:24:00	2003-09-02 07:24:00		Papa	Return to Owner	Intact Male	38.430933029336
6802	2 years	AT2805	Dog	Rotweiler	Black	2003-05-10	2003-09-23 01:13:00	2003-09-23 01:13:00		Zane	Return to Owner	Intact Male	38.466577280743
6894	2 years	AT0409	Dog	Silver Lab Husky	Black/White	2003-06-05	2003-06-02 05:41:00	2003-06-02 05:41:00		Lulu	Return to Owner	Intact Male	38.420764220275
6957	6 months	AT0561	Dog	German Shepherd	Sable	2003-07-20	2003-09-22 01:14:00	2003-09-22 01:14:00		Sargent	Return to Owner	Intact Male	38.48008959085
3138	2 years	AT2834	Dog	Silver Lab Husky	Brown/White	2004-03-05	2004-03-23 04:23:00	2004-03-23 04:23:00			Suffering	Euthanasia	Intact Male
5105	2 years	AT0702	Dog	Alaskan Malamute	Sable/White	2003-07-20	2003-09-02 07:24:00	2003-09-02 07:24:00		Papa	Return to Owner	Intact Male	38.430933029336
6802	2 years	AT2805	Dog	Rotweiler	Black	2003-05-10	2003-09-23 01:13:00	2003-09-23 01:13:00		Zane	Return to Owner	Intact Male	38.466577280743
6894	2 years	AT0409	Dog	Silver Lab Husky	Black/White	2003-06-05	2003-06-02 05:41:00	2003-06-02 05:41:00		Lulu	Return to Owner	Intact Male	38.420764220275
6957	6 months	AT0561	Dog	German Shepherd	Sable	2003-07-20	2003-09-22 01:14:00	2003-09-22 01:14:00		Sargent	Return to Owner	Intact Male	38.48008959085

Rescue Candidate Breeds



Screenshot 4 — Disaster or Individual Tracking Filter Applied

Filter returns records matching Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, and Rottweiler breeds; Intact Male; age 20–300 weeks.

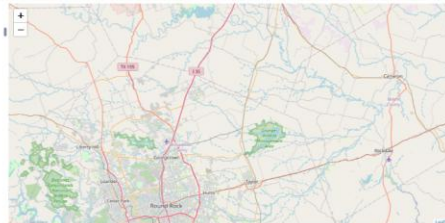
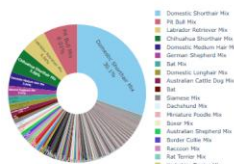


Grazioso Salvere Rescue Dog Dashboard

Developed by Nancy Ritchie - CS 460 - 55552

View Rescue <input type="radio"/> <input checked="" type="radio"/> Disaster or Wilderness Rescue <input checked="" type="radio"/> Disaster or Individual Tracking <input type="radio"/> Basic													
id	rescue_name	age_weeks_outcome	animal_id	animal_type	breed	color	date_of_birth	date_lost	month/year	name	outcome_subtype	outcome_type	sex_age_outcome
2987	4 years	A09454	Dog	Rotweiler	Black/Brown	2003-05-05	2003-04-08 04:25:00	2003-04-08 04:25:00		Striker	Return to Owner	Intact Male	38.320871209311
2967	4 years	A71226	Dog	Bloodhound	Red	2003-09-20	2003-09-22 05:43:00	2003-09-22 05:43:00		Bowser	Return to Owner	Intact Male	38.270998761287
6802	2 years	A72805	Dog	Rotweiler	Black	2003-05-10	2003-09-23 01:13:00	2003-09-23 01:13:00		Zane	Return to Owner	Intact Male	38.466577280743
6957	6 months	A70561	Dog	German Shepherd	Sable	2003-07-20	2003-09-22 01:14:00	2003-09-22 01:14:00		Sargent	Return to Owner	Intact Male	38.48008959085
2987	4 years	A09454	Dog	Rotweiler	Black/Brown	2003-05-05	2003-04-08 04:25:00	2003-04-08 04:25:00		Striker	Return to Owner	Intact Male	38.320871209311
2967	4 years	A71226	Dog	Bloodhound	Red	2003-09-20	2003-09-22 05:43:00	2003-09-22 05:43:00		Bowser	Return to Owner	Intact Male	38.270998761287
6802	2 years	A72805	Dog	Rotweiler	Black	2003-05-10	2003-09-23 01:13:00	2003-09-23 01:13:00		Zane	Return to Owner	Intact Male	38.466577280743
6957	6 months	A70561	Dog	German Shepherd	Sable	2003-07-20	2003-09-22 01:14:00	2003-09-22 01:14:00		Sargent	Return to Owner	Intact Male	38.48008959085

Rescue Candidate Breeds



4. Tools Used & Rationale

4.1 MongoDB — Model Component

MongoDB was selected as the database for several reasons that align specifically with this project's requirements:

- Document-oriented storage: Animal shelter records are naturally document-shaped (heterogeneous fields, nested data). MongoDB's BSON document model stores each animal as a self-contained record without requiring a rigid relational schema.
- Native Python integration: The PyMongo driver provides a first-class Python API. Query results are returned as Python dictionaries, which convert directly to pandas DataFrames via `pd.DataFrame.from_records()` with zero transformation overhead.
- Flexible querying: MongoDB's query language supports the `$in`, `$gte`, and `$lte` operators required by the Grazioso Salvare breed/age/sex filter specifications natively, without joins or complex SQL.
- Scalability: The 20,000-record Austin Animal Center dataset fits comfortably in MongoDB and the architecture would scale to larger shelter networks without schema changes.

4.2 Dash Framework — View & Controller

Dash (by Plotly) was chosen as the web application framework because it provides both the View and Controller layers of the MVC architecture in a single Python-native package:

- View layer: Dash's `html` and `dcc` components generate the full HTML/CSS/JS interface from Python, eliminating the need to write separate frontend code. The `dash_table.DataTable` component provides a production-ready interactive grid.
- Controller layer: Dash's `@app.callback` decorator system implements reactive data binding. Callbacks automatically re-execute when Input component values change and push results to Output components — this is the mechanism that makes the table, pie chart, and map all update simultaneously when a filter radio button is clicked.
- Plotly charts: Dash integrates natively with Plotly Express (`px.pie`) for chart generation, enabling the breed distribution pie chart to re-render from filtered `DataTable` state.
- JupyterDash: The `jupyter_dash` variant runs the Dash server inline within Jupyter/JupyterLab (as used in the Codio environment), enabling rapid development and testing without a separate server process.

4.3 Full Tool Stack

Tool / Technology	Purpose	Version
Python 3.11	Core application language	3.11.2
MongoDB	Document database (Model layer)	7.0.21
PyMongo	Python MongoDB driver	Latest
Dash / JupyterDash	Web framework (View + Controller)	Latest
Dash Leaflet	Interactive geolocation map widget	Latest
Plotly Express	Pie chart and data visualization	Latest
pandas	DataFrame manipulation and query results	Latest

Tool / Technology	Purpose	Version
dash_table	Interactive DataTable widget	Latest
base64	Logo image encoding for inline display	stdlib
Codio / JupyterLab	Cloud development environment	N/A

4.4 Resource Links

[Dash Documentation](#)

[Dash Leaflet Documentation](#)

[PyMongo Documentation](#)

[MongoDB Query Operators Reference](#)

[Plotly Express Documentation](#)

[Austin Animal Center Outcomes Dataset](#)

[Grazioso Salvare \(SNHU\)](#)

5. Steps Taken to Complete the Project

Step 1: CRUD Python Module (Project One)

Built the AnimalShelter class in CRUD_Python_Module.py implementing the four CRUD operations against the MongoDB aac.animals collection. The class accepts username and password at instantiation and connects via PyMongo's MongoClient. The read() method accepts a MongoDB query dict and returns a list of matching documents, which was the foundational interface used by all dashboard callbacks.

Step 2: Database Setup & Data Import

Imported the Austin Animal Center Outcomes CSV dataset into MongoDB using mongoimport, creating the aac database and animals collection with 20,000 records. Created the aacuser account with read/write permissions on the aac database. Verified connectivity using mongosh.

```
mongoimport --username aacuser --password SNHU12345 --authenticationDatabase admin --db aac --collection animals --type csv --headerline --file aac_shelter_outcomes.csv
```

Step 3: Dashboard Layout

Constructed the app.layout using Dash html and dcc components. Key layout decisions included: wrapping the logo in an html.A anchor tag pointing to www.snhu.edu (per spec), using dcc.RadioItems for the four filter options, configuring the DataTable with pagination/sorting/filtering/row-selection, and arranging the pie chart and map side-by-side using a flex div.

Step 4: Filter Callbacks

Implemented the `update_dashboard()` callback function, which receives the selected radio button value and constructs the appropriate MongoDB query using the exact breed lists, sex values, and age ranges from the Grazioso Salvare specification document. The filtered results are returned as a list of dicts to update the DataTable's data property.

Step 5: Chart Callbacks

Implemented `update_graphs()` to render a Plotly Express pie chart from the DataTable's `derived_virtual_data` (the post-filter, post-sort view). Implemented `update_map()` to render a Dash Leaflet map centered on the GPS coordinates of the selected DataTable row, with a popup marker showing the animal's name and breed.

Step 6: Testing & Deployment

Ran the notebook in Codio's JupyterLab environment using `app.run_server()`. Tested all four filter states (Water Rescue, Mountain Rescue, Disaster Rescue, Reset) and verified that the DataTable, pie chart, and map all updated correctly for each selection. Captured screenshots of each filter state for documentation.

6. Challenges & Solutions

Challenge 1: KeyError on `df.drop(columns=['_id'])`

The dashboard crashed on startup with a `KeyError` because the MongoDB driver version in the Codio environment did not include the `_id` field in query results by default, so there was nothing to drop.

Solution: Wrapped the drop call in an existence check: `if '_id' in df.columns: df.drop(columns=['_id'], inplace=True)`

Challenge 2: Incorrect Password — Empty DataTable

The dashboard loaded but displayed no data. The CRUD module was instantiated with password "SNHU1234" (4 digits) while the actual aacuser account password was "SNHU12345" (5 digits). MongoDB's authentication failure returned an empty list silently.

Solution: Verified credentials directly in `mongosh`, then corrected the password constant in the dashboard file.

Challenge 3: Callback Error — `selected_columns` Is None on Load

The `update_styles()` callback crashed immediately on page load because Dash fires all callbacks on startup, and `selected_columns` arrives as `None` before any user interaction. Iterating over `None` raises a `TypeError`.

Solution: Added an early return guard: `if not selected_columns: return []`

Challenge 4: Map Crash Due to Integer Column Indexing

The `update_map()` callback used `dff.iloc[row, 13]` and `dff.iloc[row, 14]` to access latitude and longitude. When the DataFrame column order changed after filtering, these integer positions pointed at wrong columns and raised `IndexError`.

Solution: Switched to named column access (`dff.iloc[row]['location_lat']`) and added a pre-flight check that all required columns exist before rendering the map.

Challenge 5: Typo `df.filtered.drop` Causing Runtime Error

A typo in the original starter code wrote `df.filtered.drop()` instead of `df_filtered.drop()`. Python interpreted `.filtered` as an attribute lookup on the DataFrame, raising `AttributeError: 'DataFrame' object has no attribute 'filtered'`. The error was masked by a broad `try/except` that fell back to the unfiltered `df`, making filters appear to have no effect.

Solution: Located the typo via `grep` and corrected `df.filtered.drop` to `df_filtered.drop`. Removed the masking fallback and replaced it with targeted error handling that prints the actual exception message to the Jupyter output.

7. Reproducing the Project

Prerequisites

- Codio account with a workspace containing JupyterLab
- Python 3.11+ with packages: `jupyter-dash`, `dash`, `dash-leaflet`, `dash[diskcache]`, `plotly`, `pandas`, `pymongo`
- MongoDB 7.0 running on `localhost:27017`
- Austin Animal Center Outcomes CSV dataset

Setup Steps

1. Start MongoDB: run `mongod --auth --bind_ip localhost` in the Codio terminal
2. Import data: run the `mongoimport` command shown in Step 2 above
3. Place `CRUD_Python_Module.py` and `Grazioso Salvare Logo.png` in the same directory as the notebook
4. Open `ProjectTwoDashboard.ipynb` in JupyterLab
5. Select Kernel > Restart & Run All
6. Click the Dash app URL printed in the output cell to open the dashboard

File Structure

```
workspace/
├── ProjectTwoDashboard.ipynb      # Main dashboard notebook
├── CRUD_Python_Module.py         # AnimalShelter CRUD class
```

```
|— Grazioso Salvare Logo.png      # Company logo (branding)
|— aac_shelter_outcomes.csv        # Source data for import
```