**Grazioso Salvare Dashboard — CS 340 README**

# **About the Grazioso Salvare Dashboard**

Grazioso Salvare — CS-340 Dashboard is a Dash web app that queries a MongoDB database of Austin Animal Center Outcomes to help trainers quickly find dogs suited for Water Rescue, Mountain/Wilderness Rescue, and Disaster/Individual Tracking.

**The app presents:**

* an interactive filter for rescue type,
* a DataTable (sort, filter, paginate, row-select),
* a Breed Distribution donut chart
* a Geolocation Map centered on the selected animal (or clustered markers when none is selected).

***Note:*** *Branding requirements are met by displaying the Grazioso Salvare logo and Unique identifier*

# **Motivation**

Grazioso Salvare needs a streamlined, low-training interface to identify candidate dogs by rescue profile. This dashboard reduces manual querying, standardizes selection criteria, and provides visual confirmation (distribution + location) to support decisions.

# **Getting Started**

*To run this locally in your computer, you need the following:*

## **Installation (tools you need)**

* Ensure Python, Jupyter, and MongoDB are available.
* Install dependencies and python packages with the pip command:

**pip install jupyter-dash dash==2.\* dash-leaflet plotly pandas numpy pymongo**

## **Files Needed (keep in the same folder)**

* ProjectTwoDashboard.ipynb — the dashboard (Project Two)
* animal\_shelter.py — CRUD module (Project One)
* Grazioso Salvare Logo.png — branding image
* aac\_shelter\_outcomes.csv — dataset used to populate AAC.animals

## **Importing / Loading the Database (one time setup)**

// start mongosh,then:

**use AAC**

**db.createUser({ user:"aacuser", pwd:"youruniquepassword", roles:[{ role:"readWrite", db:"AAC" }] })**

# mongoimport (local)

**mongoimport --db AAC --collection animals --type csv --headerline --drop\**

**--username aacuser --password youruniquepassword --authenticationDatabase AAC\**

**aac\_shelter\_outcomes.csv**

## **How to Run**

* **Apporto/Jupyter notebook (recommended):** Open ProjectTwoDashboard.ipynb → **Restart & Run All**.  
  The app renders **inline** and also provides **“Open in new tab ↗”** at the top.
* **Local:** Start MongoDB → open the notebook in Jupyter → **Run All**.

# **Usage**

1. Choose **Rescue Type**: Reset, Water Rescue, Mountain/Wilderness, or Disaster/Tracking.
2. Review and interact with the **DataTable** (sort, filter, paginate, select a row).
3. Inspect the **donut chart** (auto-updates to current table rows; top-N + “Other”).
4. View the **map**: select a row to center on that animal; otherwise see clustered markers.

## **Code Example**

# Map each rescue type to a MongoDB query

**RESCUE\_QUERIES = {**

**"Water Rescue": {**

**"animal\_type": "Dog",**

**"breed": {"$in": ["Labrador Retriever Mix","Chesapeake Bay Retriever","Newfoundland"]},**

**"sex\_upon\_outcome": "Intact Female",**

**"age\_upon\_outcome\_in\_weeks": {"$gte": 26, "$lte": 156},**

**},**

**"Mountain or Wilderness Rescue": {**

**"animal\_type": "Dog",**

**"breed": {"$in": ["German Shepherd","Alaskan Malamute","Old English Sheepdog","Siberian Husky","Rottweiler"]},**

**"sex\_upon\_outcome": "Intact Male",**

**"age\_upon\_outcome\_in\_weeks": {"$gte": 26, "$lte": 156},**

**},**

**"Disaster or Individual Tracking": {**

**"animal\_type": "Dog",**

**"breed": {"$in": ["Doberman Pinscher","German Shepherd","Golden Retriever","Bloodhound","Rottweiler"]},**

**"sex\_upon\_outcome": "Intact Male",**

**"age\_upon\_outcome\_in\_weeks": {"$gte": 20, "$lte": 300},**

**},**

**}**

# Controller: filter -> table

**@app.callback(Output("datatable-id","data"),Input("filter-type","value"))**

**def update\_dashboard(filter\_name):**

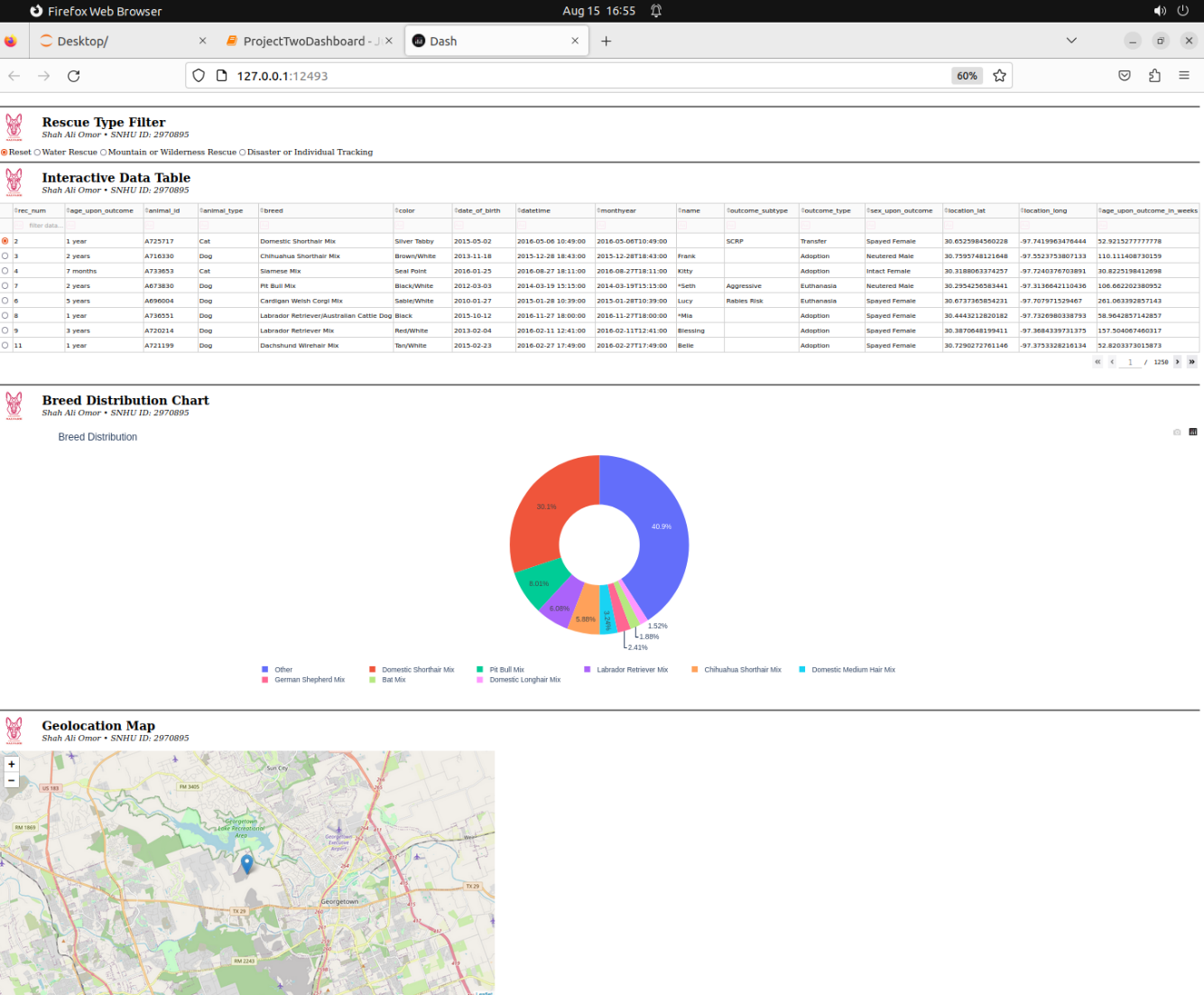
**return fetch\_df(filter\_name).to\_dict("records")**

## **Tests**

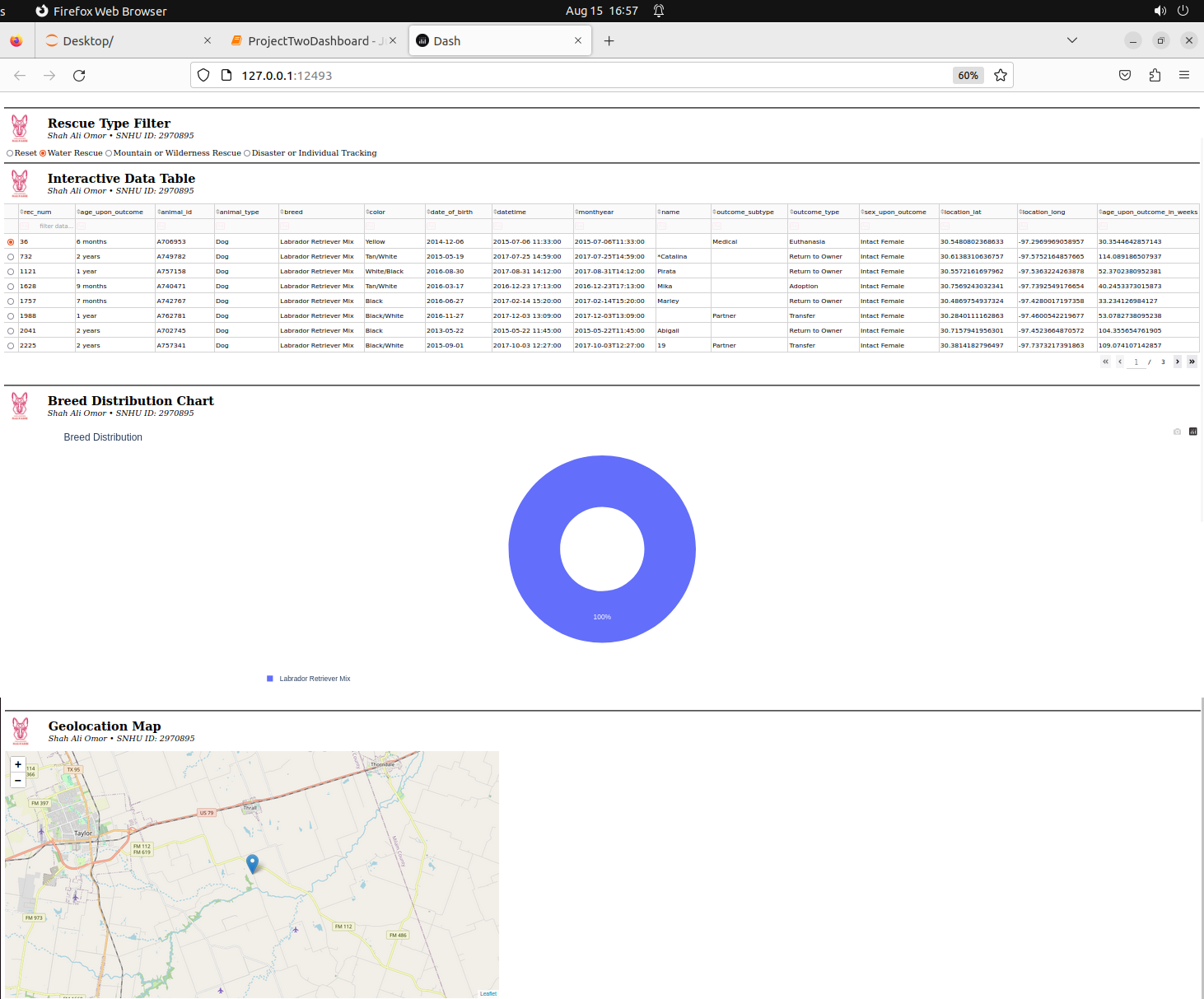
* Change the rescue filter and confirm the table, chart, and map all update.
* Select different rows and confirm the map rec enters correctly.
* Optional unit checks for CRUD (create→read→update→read→delete) against a test record.

## **Screenshots**

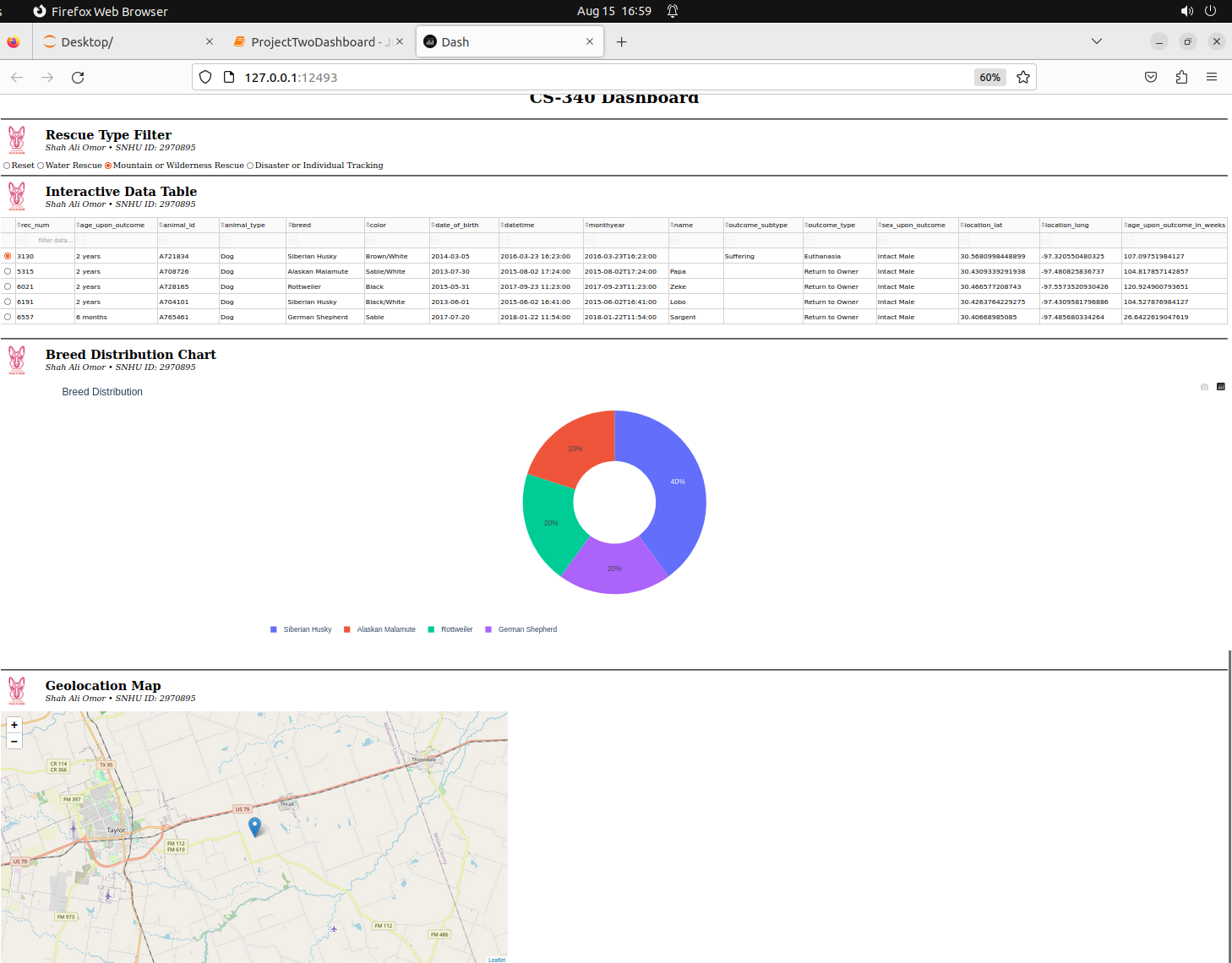
### **Starting State / Reset**



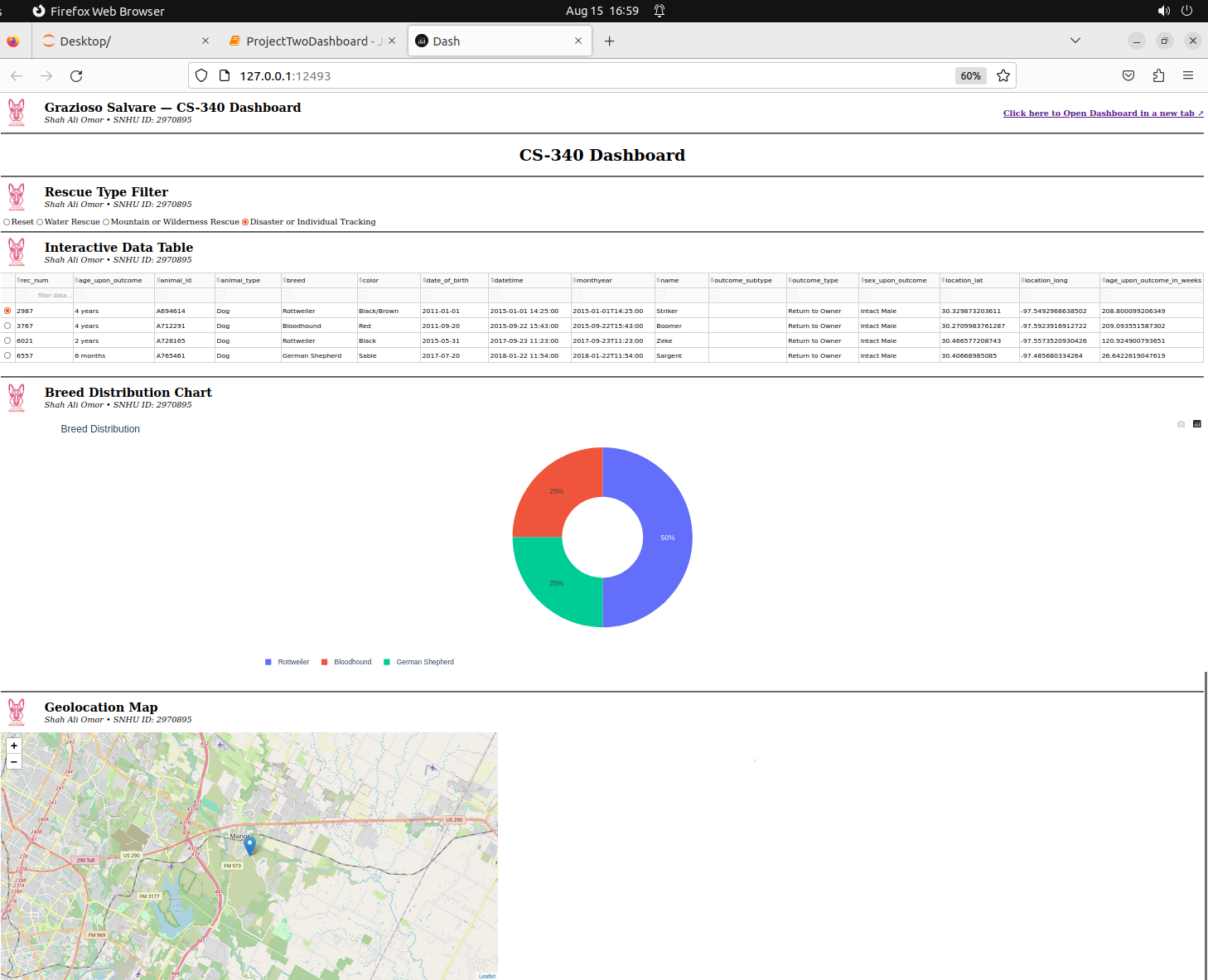
### **Water Rescue**



### **Mountain or Wilderness Rescue**



### **Disaster or Individual Tracking**



# **Required Functionality**

* **Architecture (MVC):** MongoDB + animal\_shelter.py (Model), Dash callbacks (Controller), Dash components (View).
* **Startup:** Loads unfiltered data; drops \_id for table compatibility.
* **Filters:** Radio items → Reset, Water, Mountain/Wilderness, Disaster/Tracking; each runs the correct MongoDB query (breed/sex/age).
* **DataTable:** Sort, filter, paginate, single-row select; refreshes on filter change.
* **Charts:** Donut chart updates from current table rows (top-N + “Other”); Map centers on selected row or shows clustered markers.
* **Branding & Evidence:** Logo + identifier visible; four screenshots (Reset, Water, Mountain/Wilderness, Disaster/Tracking).

# **Tools Used & Rationale**

* **MongoDB + PyMongo:** JSON-like documents match the CSV well; flexible schema; expressive filters ($in, range) used by rescue profiles; simple Python integration.
* **Dash/JupyterDash:** View (components) and Controller (callbacks) in pure Python; renders inline in the VM and supports a full-page view link.
* **Plotly:** Interactive donut chart with readable labels/legend; perfect for quick distribution insight.
* **dash-leaflet:** Lightweight, tile-based map; marker clustering keeps the map useful even with many points.

# **Steps Taken To Complete The Project**

1. Loaded data: Imported aac\_shelter\_outcomes.csv into MongoDB as AAC.animals and created the user aacuser/omor123.
2. Built backend: Wrote animal\_shelter.py with create/read/update/delete and verified with a test record.
3. Started app: Opened ProjectTwoDashboard.ipynb in Jupyter (Apporto), connected to MongoDB through the CRUD class, fetched an unfiltered dataset, dropped \_id, and coerced lat/long to numeric.
4. Added branding: Placed the Grazioso Salvare logo and “Shah Ali Omor • SNHU ID: 2970895” above each section and added an Open in new tab link.
5. Defined filters: Created RESCUE\_QUERIES for Water, Mountain/Wilderness, and Disaster/Tracking with the exact breed, sex, and age rules.
6. Wired callbacks:
   * Filter → query MongoDB → refresh DataTable
   * Table view → update donut chart
   * Selected row (or none) → update map (center on row or show clustered markers)
7. Polished table: Enabled sorting, filtering, pagination, and single-row selection for usability.
8. Tested & captured evidence: Ran end-to-end and saved four screenshots: Reset, Water, Mountain/Wilderness, Disaster/Tracking (each shows filter + table + chart + map + branding).

# **Challenges & How They Were Overcome**

* ObjectId in the table: Dash couldn’t serialize Mongo’s \_id; removed the \_id column before rendering to keep the table stable.
* Bad/missing coordinates: Some rows lacked valid lat/long; coerced to numeric and, if a selected row has no coords, showed a clustered map of all valid points.
* Age-range query bug: Separate $gte/$lte blocks overwrite each other; combined them in one field (e.g., {"$gte":26,"$lte":156}) to enforce the range.
* Crowded chart labels: Many breeds made the pie unreadable; used a large donut, kept top-N, and grouped the rest as “Other.”
* Branding in every screenshot: To avoid missing the logo/ID, added a reusable header above each section.
* Noisy server logs: Access logs cluttered the notebook; silenced route logging to keep outputs clean.

# **Resources**

* **Data Set:** Austin Animal Center. (2020). *Austin Animal Center Outcomes* [Data set]. City of Austin, Texas Open Data Portal. <https://doi.org/10.26000/025.000001>
* **Image:** *Grazioso Salvare Logo*`

# **Contact**

**Shah Ali Omor** — [omoranam@gmail.com](mailto:omoranam@gmail.com)