**CS-340 README Grazioso Salvare’s AG Dashboard**

**The Project**

The purpose of this project was to create a dashboard for Grazioso Salvare, allowing them to browse through the animals that are currently housed at the Austin Animal Shelter that align with their business requirements.

**Dashboard layers**

The MongoDB Layer holds the data for visualization purposes.

The Python Middleware Layer is utilized for control functions.

The Plotly Dash Leaflet Visualization Layer generates the HTML content for the dashboard.

The Python middleware utilizes a class that helps with performing Create, Read, Update, and Delete (CRUD) tasks in MongoDB using the PyMongo driver. This class includes all the abstracted getter/setter methods for CRUD actions and initiating MongoDB.

**Components Selected**

**MongoDB**

We chose MongoDB over SQL due to its ease of use and greater adaptability. Its dynamic schema reduces the likelihood of errors when updating or creating data, like inputting a decimal for an age intended to be an integer. Moreover, MongoDB's JSON format is well suited for web settings.

**Python**

Python offers a variety of resources for linking the backend and frontend. One such tool is the PyMongo driver, which facilitates interaction with MongoDB through the CRUD class and middleware layer. Additionally, Python allows for the creation of HTML/JavaScript dashboards and widgets using libraries like Dash and Dash Leaflet.

**Plotly-Dash and Dash Leaflet**

These libraries act as interfaces for tools based on JavaScript that communicate with Python using their APIs. They enable the creation of an SVG (for pie charts), a tiled PNG (for maps), or the generated  HTML code and CSS to display data tables and widgets.

**CRUD Class**

Methods used:

Constructor, sets up the class using MongoDB login details.

createRecord(data), for inserting a new record.

deleteRecord(query), used to delete records matching the query.

getRecordId(data), sets up command to retrieve a record by its ID.

getRecordCriteria(criteria=None), used to retrieve records matching the criteria.

updateRecord(query, newValue), method used to update records matching the query with new values

**Properties used:**

records\_updated

records\_matched

records\_deleted

**Pre-Conditions/ Installation**

Python 3.6 or later

PyMongo 4.2 [pymongo Link](https://pymongo.readthedocs.io/en/stable/installation.html)

MongoDB 4.2 [Mongodb link](https://docs.mongodb.com/manual/installation/)

Plotly Dash [ploty link](https://dash.plotly.com/installation)

Dash Leaflet [dash leaflet](https://www.dash-leaflet.com/)

Importing csv database collection into MongoDB : [link](https://www.mongodb.com/docs/compass/current/import-export/)

User account with read and write permissions

**Dashboard Walk-though**

Grazioso Salvare Requirements:

Branding done with logo

A table of data, with filters that shows details about the animals housed at the Austin Animal Shelter.

These custom filters help pinpoint animals based on purposes, like water rescue, mountain or wilderness rescue and disaster rescue or individual tracking.

Pie chart with a breakdown of available breeds.

Interactive Map showing the location of animals.

**Must have features include:**

Client Logo

Custom Filters

Dynamic Data Table

Dynamic Pie Chart

Map

Toggle to control map

Inline filters for Sorting

**Filter Specifications**

Once a filter option is chosen the dynamic data table, the map showing locations and the additional chart will all reflect the selected data.

Water Rescue filter, looks for a mix of Labrador Retriever, Chesapeake Bay Retriever and Newfoundland; Female dog that has not been spayed; Aged, between 6 months to 3 years.

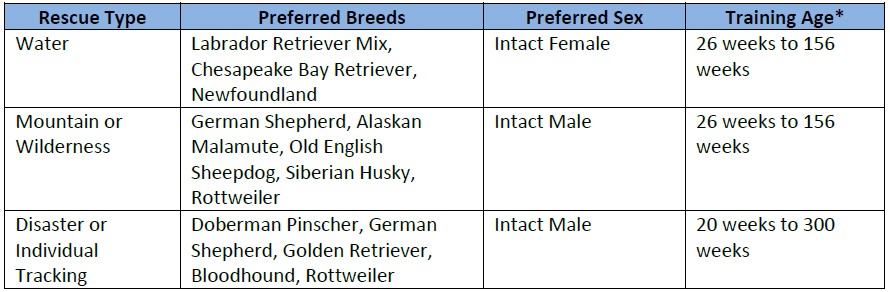
Mountain or wilderness rescue filter looks for dogs, like German Shepherds, Alaskan Malamutes, Old English Sheepdogs, Siberian Huskies and Rottweilers. Prefer males aged between 26 weeks and 156 weeks.

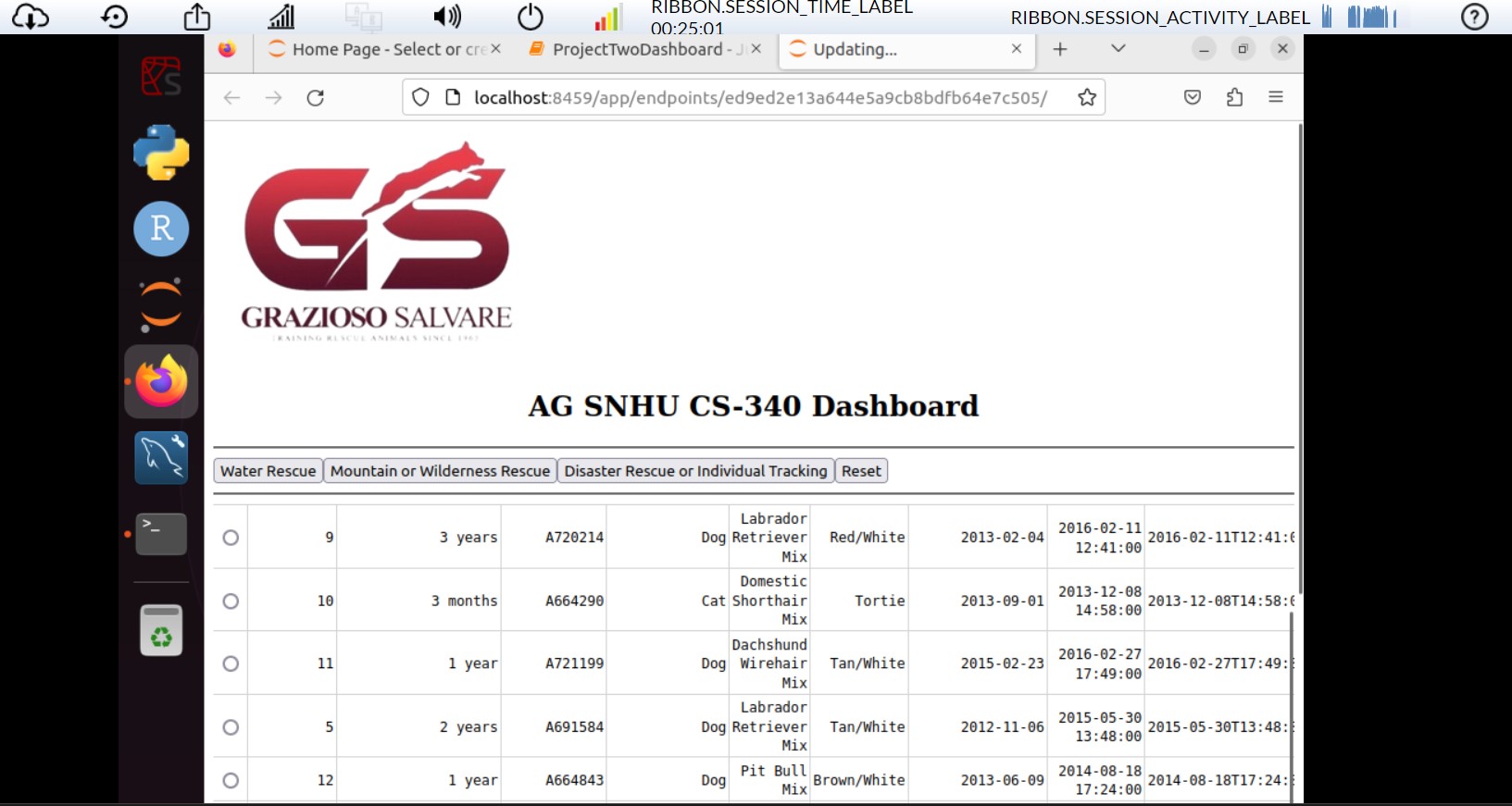
Individual Tracking Filter that searches for dog breeds, like Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound and Rottweiler; Intact Male; aged, between 20 weeks and 300 weeks.

**Dashboard Recreation**

1. Install components required see (Preconditions/Installation)
2. Use admin account to load database into MongoDB
3. Must create the needed user roles (read/write)for CRUD class.
4. Time to add the CRUD class and middleware driver to host server
5. Middleware dashboard must have user credentials which were created in step 3.
6. Now add callbacks which I have attached in (Appendix) and dash components as well.
7. Now its time to update regular expressions for filtering.
8. Start up MongoDB
9. And start Python middleware dashboard.

**Client requirements :**



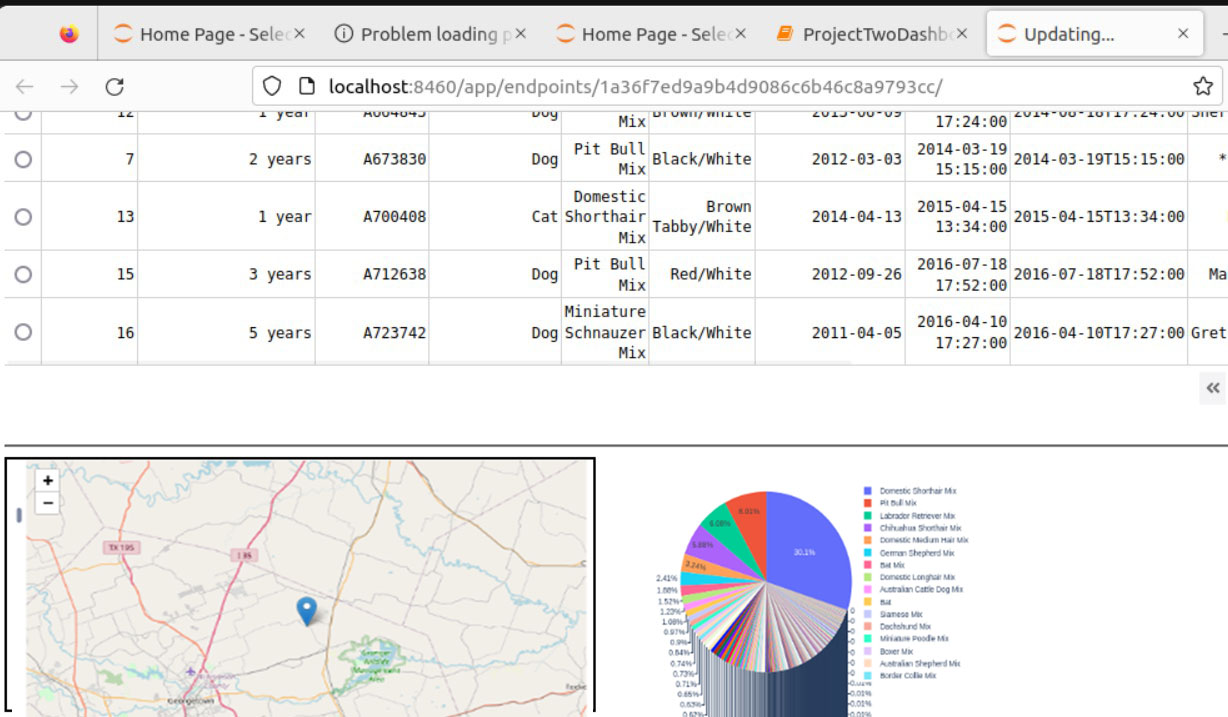


Toggle buttons for the map

Inline filters

Required filters for searches

🡨 GS logo which links to their website



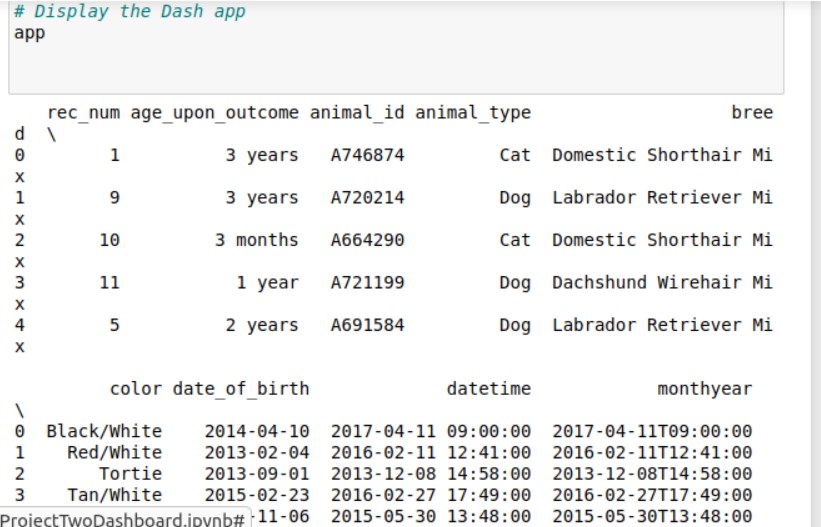
Map with location of animal queried

Pie chart which updates with chosen query

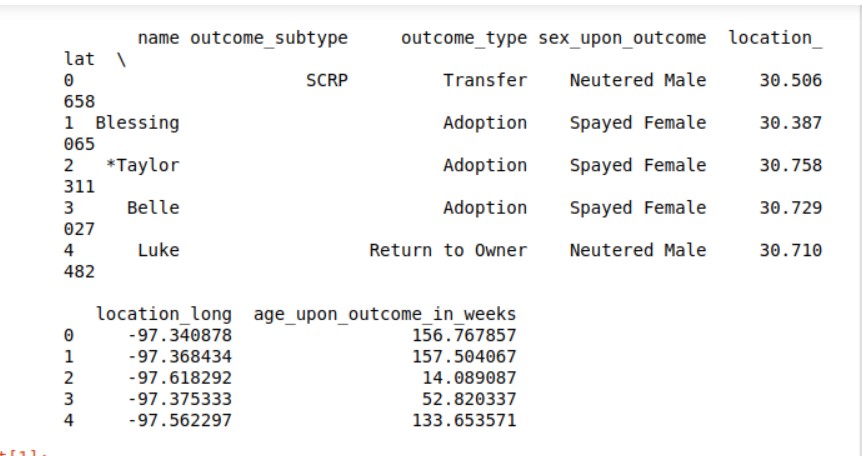
**Struggles encountered**

The primary challenge involved sorting through the information, from the Austin Animal Shelter to fit criteria. The database contained data, which called for using expressions, for matching patterns. PyMongo doesn't naturally support expressions specific formatting was necessary for the PyMongo API to recognize them. The interactive map wasn't displaying as expected prompting me to add debugging functions to ensure its functionality. See (Debug example 1 and 2 )

**Debug Example 1**



**Debug Example 2**



**Contact**

**Armando Gomez ,** [**armando.gomez@snhu.edu**](mailto:armando.gomez@snhu.edu) **, created for CS-340**

**Appendix**

**Dash Callbacks**

**#############################################**

**# Interaction Between Components / Controller**

**#############################################**

**@app.callback(**

**[Output('datatable-id', 'data'),**

**Output('button-output', 'children'),**

**Output('graph-id', "children")],**

**[Input('button-one', 'n\_clicks'), Input('button-two', 'n\_clicks'),**

**Input('button-three', 'n\_clicks'), Input('button-four', 'n\_clicks')]**

**)**

**def update\_dashboard(button1, button2, button3, button4):**

**ctx = dash.callback\_context**

**if not ctx.triggered:**

**button\_id = 'No clicks yet'**

**else:**

**button\_id = ctx.triggered[0]['prop\_id'].split('.')[0]**

**button\_output = "Displaying all data"**

**if button\_id == 'button-one':**

**data = pd.DataFrame.from\_records(shelter.getRecordCriteria({'$and': [**

**{'$or': [{'breed': 'Labrador Retriever Mix'}, {'breed': 'Chesapeake Bay Retriever'}, {'breed': 'Newfoundland'}]},**

**{'sex\_upon\_outcome': 'Intact Female'}, {'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}}**

**]}))**

**button\_output = "Filtering data for Water Rescue"**

**elif button\_id == 'button-two':**

**data = pd.DataFrame.from\_records(shelter.getRecordCriteria({'$and': [**

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

**{'sex\_upon\_outcome': 'Intact Male'}, {'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}}**

**]}))**

**button\_output = "Filtering data for Mountain or Wilderness Rescue"**

**elif button\_id == 'button-three':**

**data = pd.DataFrame.from\_records(shelter.getRecordCriteria({'$and': [**

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

**{'sex\_upon\_outcome': 'Intact Male'}, {'age\_upon\_outcome\_in\_weeks': {'$gte': 20, '$lte': 300}}**

**]}))**

**button\_output = "Filtering data for Disaster Rescue or Individual Tracking"**

**else:**

**data = pd.DataFrame.from\_records(shelter.getRecordCriteria({}))**

**button\_output = "Displaying all data"**

**if '\_id' in data.columns:**

**data.drop(columns=['\_id'], inplace=True)**

**# Update the pie chart**

**chart = [**

**dcc.Graph(**

**figure=px.pie(data, names='breed', title='Percentage of breeds available')**

**)**

**]**

**return data.to\_dict('records'), button\_output, chart**

**@app.callback(**

**Output('datatable-id', 'style\_data\_conditional'),**

**[Input('datatable-id', 'selected\_columns')]**

**)**

**def update\_styles**