CS-340 README for Grazioso Salvare’s Custom Dashboard

## About the Project

This project was conducted to build a functional, web-based, dashboard for Grazioso Salvare so they may review animals available at the Austin Animal Shelter that fit their business needs.

This project is a custom-built dashboard for Grazioso Salvare, an animal rescue organization. The dashboard provides a user-friendly interface for filtering, viewing, and analyzing animal data stored in a MongoDB database. It includes interactive charts and a geolocation map, all powered by Python, Dash, and Plotly.

The dashboard provides the following functionality:

- \*\*Filter by Rescue Type\*\*: Users can select from predefined filters (All, Water Rescue, Mountain Rescue, Disaster Rescue) to retrieve animals that meet specific breed and age criteria.

- \*\*Data Table View\*\*: View a searchable, sortable, and paginated table of animal data, with column highlighting.

- \*\*Interactive Pie Chart\*\*: Visualize the breed distribution of the currently displayed data.

- \*\*Geolocation Mapping\*\*: Select an animal to view its approximate intake location on a map centered around Austin, Texas.

- \*\*Organization Branding\*\*: Includes the Grazioso Salvare logo linked to SNHU's website.

## Component Selection Methodology

MongoDB was selected instead of a traditional SQL database due to its user-friendly design and adaptable data structure. Its schema-less architecture allows for greater flexibility during data insertion and updates, minimizing the likelihood of errors for instance, accepting decimal values for age without issues that a rigidly typed SQL schema might cause. Additionally, MongoDB stores data in a JSON-like format, which naturally aligns with modern web application structures and APIs.

Although MongoDB does not provide the same strict ACID compliance as SQL databases, the scope of this project does not involve complex or high-volume transactional operations, so this trade-off is acceptable.

Python was the ideal language for this project thanks to its extensive ecosystem for both backend and frontend development. The pymongo library enables seamless integration with MongoDB, facilitating CRUD operations through a dedicated abstraction layer. On the frontend side, Python-based frameworks like Plotly Dash and Dash Leaflet serve as powerful tools to build interactive visual dashboards. These libraries act as Python wrappers around JavaScript technologies, allowing developers to create components such as SVG-based pie charts, tile-based maps, and HTML tables with responsive styling without having to write JavaScript directly.

## CRUD Class

CRUD operations are commonplace when using any database but formatting these queries and any error handling can be tedious. To help alleviate the tedium of formatting each new query or re-code for each error type these operations have been abstracted to simple get and set methods.

The class contains the following methods:

* Constructor (password, username)
* createRecord(data)
* deleteRecord (query)
* getRecordId(data)
* getRecordCriteria(criteria = None)
* updateRecord (query, newValue)

And properties:

* records\_updated
* records\_matched
* records\_deleted

For more information on the CRUD class refer to the specific readme.

**Installation/Pre-Conditions**

To use this software the following tools must be installed:

* Python 3.6
* Pymongo 4.2: [link](https://pymongo.readthedocs.io/en/stable/)
* MongoDB 4.2: [link](https://www.mongodb.com/)
* Plotly Dash: [link](https://dash.plotly.com/)
* Dash Leaflet: [link](https://dash-leaflet.herokuapp.com/)
* Data available in a Mongo database collection, see example on importing a CSV
* User account with read/write permissions

## Product Walkthrough

Grazioso Salvare requested the dashboard to have a few key components:

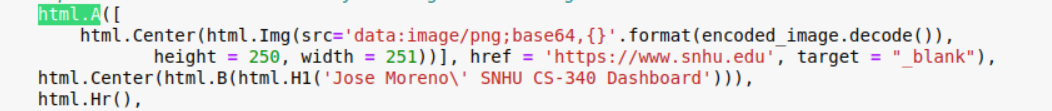
* Branding with logo

A logo of a dog

AI-generated content may be incorrect.

* A link to their website

The link is imbedded into the image.



* A data table, with custom filters, containing the information about the animals at the Austin Animal Shelter
  + Custom filters provide targeted animals for specific business needs, including:
    - Water Rescue
    - Mountain or Wilderness Rescue
    - Disaster Rescue or Individual Tracking
    - A pie chart showing the breakdown of available breeds
    - A map with the location of a selected animal

A screenshot of a computer

AI-generated content may be incorrect.

The custom filters are set with the specifications provided by the client, see below.

A screenshot of a computer

AI-generated content may be incorrect.

If a user chooses a new filter the pie chart and data table are updated, see below.

A screenshot of a map

AI-generated content may be incorrect.

If a particular animal is selected its location is shown on the map, otherwise the Austin Animal Shelter is shown, see below.

A screenshot of a map

AI-generated content may be incorrect.

If the map marker is clicked the animal’s name is show and the breed is shown on hover.

A map with a blue point on it

AI-generated content may be incorrect.

## Project Recreation

If this project needed to be recreated due to server migration or failed back up the following steps can be followed.

1. Install the required components, see section *Installation/Pre-Conditions*
2. Use an admin account to load data into MongoDB, *mongoimport* is suggested  
   A screen shot of a computer code

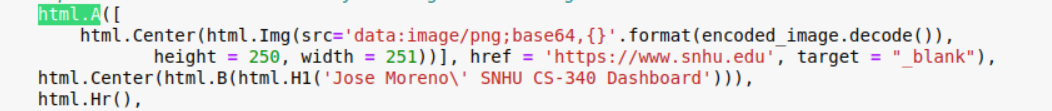
   AI-generated content may be incorrect.
3. Create the needed user roles (CRUD class does not need full admin rights just read/write) – two accounts (admin and a user role) are shown below  
   A computer screen shot of a computer program

   AI-generated content may be incorrect.
4. Add the CRUD class and the middleware-dashboard driver to the host server
5. Update the middleware-dashboard driver with username and password created in **Step 3**  
   A computer screen shot of a login code

   AI-generated content may be incorrect.
6. Add the Dash components and their callbacks (see Appendix) as needed
   * Data table code example  
     A screen shot of a computer

     AI-generated content may be incorrect.
   * Html div holding widgets example  
     A screen shot of a computer

     AI-generated content may be incorrect.
   * Radio buttons acting as filters example  
       
     A close-up of a computer code

     AI-generated content may be incorrect.
   * Html anchor for logo and link to homepage example  
     
7. Update any regular expressions used for filtering if they have changed, *Water Rescue* is shown  
   A screen shot of a computer code

   AI-generated content may be incorrect.
8. Start MongoDB
9. Start the Python middleware-dashboard driver

## Challenges And Resolutions

Issue: MongoDB doesn’t support $in with regex directly.

Solution: Used $or with multiple $regex conditions.

Issue: Latitude and longitude were stored as strings.

Solution: Converted using float() and .to\_string().split()[1] for parsing.

Issue: Static assets (like the logo) were not displaying.

Solution: Encoded image using base64 to embed it directly in HTML.

**Contact**

Jose Moreno, [Jose.moreno4@snhu.edu](mailto:Jose.moreno4@snhu.edu), created for SNHU CS-340 on June 22nd, 2025

## Appendix

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

*# Interaction Between Components / Controller*

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

*@app.callback([Output('datatable-id','data'),*

*Output('datatable-id','columns')],*

*[Input('filter-type', 'value')])*

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

*#set up an if/else if/else block to respond to the radio buttons*

*if filter\_type == 'All':*

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

*elif filter\_type == 'Water':*

*#data isn't that clean, use regex for pattern matching*

*#build the regex patterns for the different filters*

*labRegex = re.compile(".\*lab.\*", re.IGNORECASE)*

*chesaRegex = re.compile(".\*chesa.\*", re.IGNORECASE)*

*newRegex = re.compile(".\*newf.\*", re.IGNORECASE)*

*df = pd.DataFrame.from\_records(shelter.getRecordCriteria({*

*'$or':[ #Regex isn't allowed in an $in helper so use $or*

*{"breed": {'$regex': newRegex}}, #pass the regex to the filter*

*{"breed": {'$regex': chesaRegex}},*

*{"breed": {'$regex': labRegex}},*

*],*

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

*"age\_upon\_outcome\_in\_weeks": {"$gte":26.0, "$lte":156.0}*

*}))*

*elif filter\_type == 'Mountain':*

*germanRegex = re.compile(".\*german.\*", re.IGNORECASE)*

*alaskanRegex = re.compile(".\*mala.\*", re.IGNORECASE)*

*oldRegex = re.compile(".\*old engilish.\*", re.IGNORECASE)*

*huskyRegex = re.compile(".\*husk.\*", re.IGNORECASE)*

*rottRegex = re.compile(".\*rott.\*", re.IGNORECASE)*

*df = pd.DataFrame.from\_records(shelter.getRecordCriteria({*

*'$or':[*

*{"breed": {'$regex': germanRegex}},*

*{"breed": {'$regex': alaskanRegex}},*

*{"breed": {'$regex': oldRegex}},*

*{"breed": {'$regex': huskyRegex}},*

*{"breed": {'$regex': rottRegex}},*

*],*

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

*"age\_upon\_outcome\_in\_weeks": {"$gte":26.0, "$lte":156.0}*

*}))*

*elif filter\_type == 'Disaster':*

*germanRegex = re.compile(".\*german.\*", re.IGNORECASE)*

*goldenRegex = re.compile(".\*golden.\*", re.IGNORECASE)*

*bloodRegex = re.compile(".\*blood.\*", re.IGNORECASE)*

*doberRegex = re.compile(".\*dober.\*", re.IGNORECASE)*

*rottRegex = re.compile(".\*rott.\*", re.IGNORECASE)*

*df = pd.DataFrame.from\_records(shelter.getRecordCriteria({*

*'$or':[*

*{"breed": {'$regex': germanRegex}},*

*{"breed": {'$regex': goldenRegex}},*

*{"breed": {'$regex': bloodRegex}},*

*{"breed": {'$regex': doberRegex}},*

*{"breed": {'$regex': rottRegex}},*

*],*

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

*"age\_upon\_outcome\_in\_weeks": {"$gte":20.0, "$lte":300.0}*

*}))*

*else:*

*raise Exception("Unknown filter")*

*columns=[{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns]*

*data=df.to\_dict('records')*

*return (data,columns)*

*#change the color of a selected cell*

*@app.callback(*

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

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

*)*

*def update\_styles(selected\_columns):*

*return [{*

*'if': { 'column\_id': i },*

*'background\_color': '#D2F3FF'*

*} for i in selected\_columns]*

*#call back for pie chart*

*#set to plot all of the data across all of the pages instead of the viewable data*

*#change to derived\_viewport\_data if other behavior is wanted*

*@app.callback(*

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

*[Input('datatable-id', "derived\_virtual\_data")])*

*def update\_graphs(viewData):*

*dffPie = pd.DataFrame.from\_dict(viewData)*

*return [*

*dcc.Graph(*

*figure = px.pie(dffPie, names='breed',)*

*)*

*]*

*#call back for slecting a row and then plotting the geomarker*

*@app.callback(*

*Output('map-id', "children"),*

*[Input('datatable-id', "derived\_virtual\_selected\_rows")])*

*def update\_map(virtualRows):*

*#austin Texas is [30.75, -97.48]*

*#create the views*

*if not virtualRows: #build a default view if there are no selected lines*

*markerArray = (30.75,-97.48) #default marker at Austin Animal Shelter*

*toolTip = "Austin Animal Center"*

*popUpHeading = "Austin Animal Center"*

*popUpParagraph = "Shelter Home Location"*

*else: #build the contextual views based on the selection*

*dff = pd.DataFrame(df.iloc[virtualRows]) #convert the datatable to a dataframe*

*coordLat = float(dff['location\_lat'].to\_string().split()[1]) #strip out the lat*

*coordLong = float(dff['location\_long'].to\_string().split()[1]) #strip out the long*

*markerArray = (coordLat, coordLong) #build the array based on selection*

*toolTip = dff['breed']*

*popUpHeading = "Animal Name"*

*popUpParagraph = dff['name']*

*#return the map with a child marker*

*#marker is set to the values found in markerArray*

*#map centers/moves to view the new marker instead of holding a fixed center*

*return [dl.Map(style={'width': '700px', 'height': '450px'}, center=markerArray,*

*zoom=10, children=[dl.TileLayer(id="base-layer-id"),*

*dl.Marker(position=markerArray, children=[*

*dl.Tooltip(toolTip),*

*dl.Popup([*

*html.H1(popUpHeading),*

*html.P(popUpParagraph)*

*])*

*])*

*])*

*]*