# CS 340 README Template

## About the Project/Project Title

This project was made for building a functional web-based dashboard for the rescue-animal training company Grazioso Salvare. This web-based dashboard will help the company Grazioso Salvare review animals available at the Austin Animal Shelter to locate dogs that can be used in rescue training.

This project contains three layers:

* A MongoDB layer in which this layer contains the data needed for visualization.
* A Python middle-ware layer in which it is used for control.
* A Plotly-, Dash-, and Leaflet-based visualization layer in which it is used to produce the html.

The Python middleware uses a custom class that helps facilitate the Create-Read-Update-Delete (CRUD) actions within MongoDB via the use of the Pymongo driver. This class contains all the abstract getter/setter methods needed for the CRUD operations and MongoDB initiation.

## Motivation

MongoDB was chosen over SQL specifically for the ease of use and for the flexibility that it provides. MongoDB utilizes a dynamic schema in which it helps prevent errors from occurring during an update or create operation, such as an individual entering an age with decimals instead of an integer. In addition, the JSON format from MongoDB is best suited for a web environment. However, MongoDB does not guarantee the same level of ACID properties as with the likes of SQL, that being said this tool was not expected to have many transactions therefore the issue was disregarded. Python was chosen to be used within this project as it has many tools available to connect the backend and front end. For instance, the Pymongo driver used in the project is to interact with MongoDB via the CRUD class as well as the middleware layer. Python is used to build html/JavaScript based dashboards as well as other widgets via the use of Plotly-Dash, and Dash Leaflet libraries. Plotly-Dash, and Dash Leaflet libraries serve as wrappers around JavaScript based tools that interact with Python through the use of their respective APIs. These libraries help make it possible to generate the SVG (pie chart), the tiled-PNG (map), or generate the raw html code and CSS used to render the data table and widgets used in this project.

## CRUD Class

The CRUD operations are quite common when using any database, however formatting these queries as well as any error handling can be very tedious. To alleviate the tedium of formatting each new query/re-code for each error type, these operations were abstracted to simple get and set methods.

The CRUD class contains the following methods:

* Constructor (username, password)
* createRecord(data)
* deleteRecord(query)
* getRecordId(data)
* getRecordCriteria(criteria = None)
* updateRecord(query, newValue)
* Following properties are available within the project’s class to help in understanding recent interactions
  + records\_updated
  + records\_matched
  + records\_deleted

## Installation and Pre-Conditions

To use this abstraction the following tools must be installed:

* Python 3.9
* MongoDB 6.0.13
* Pymongo 3.12.0
* Plotly Dash
* Dash Leaflet
* Data available through MongoDB collection
* User account with permissions

## Walkthrough

## For this project Grazioso Salvare requested the dashboard contain the following key components:

* Branding with logo
* Link to the website
* Data table with custom filters, as well as information pertaining to the animals at the Austin Animal Shelter
  + Custom filters that targeted specific animals
    - Water Rescue
    - Mountain or Wilderness Rescue
    - Disaster Rescue or Individual Training
* Pie chart (or other graph) providing breakdown of available breeds at the shelter
* A map with the location of the selected animal.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

As shown in the image above the custom filters are set with the specifications provided by the client. If a user chooses a new filter, the provided chart along with the data table will update to reflect the changes made by the user’s actions.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

As shown in the images above, if a particular animal is selected the location of that animal will appear on the provided map along with a tooltip with the name of the animal appearing if the marker is clicked. If no animal is selected, then the default marker will be the location of the animal shelter. Furthermore, it is not shown in the images above, however, should the user hover over the marker once an animal has been selected that animal’s breed will display until the cursor moves away from the marker.

## Project Recreation

Should this project were to or needed to be recreated due to server migration or because of a failed backup, then the following steps can be followed bellow:

1. Install required components (see Installation section)
2. Use admin account to load data into MongoDB (using mongoimport in MongoDB is suggested)

A screenshot of a computer program

Description automatically generated

1. Create needed user roles (CRUD class does not require full admin rights, just read/write), shown below are the admin and user roles

A screenshot of a computer screen

Description automatically generated

1. Add CRUD class and middleware-dashboard driver to host server
2. Update middleware-dashboard driver with username and password created in step 3

A computer screen shot of a computer code

Description automatically generated

1. Add Dash components and their callbacks (see Appendix below) as needed

* Data table code example

A screen shot of a computer

Description automatically generated

* + Html div holding widgets example

A white screen with colorful text

Description automatically generated

* + Radio buttons acting as filters example

A screen shot of a computer

Description automatically generated

* + Html anchor example

A close-up of a computer screen

Description automatically generated

1. Update any regular expressions used for filtering if changed (example below)



## Start MongoDB

1. Start Python middleware-dashboard driver

## Pit Falls and Struggles

For individuals trying to recreate or replicate the project, the main struggle while working on this project was filtering the Austin Animal Shelter data. Grazioso Salvare had specific requirements on the breeds of dogs, however, the data provided in the database was not very clean. An example of this can be seen with Grazioso Salvare’s interests with the Chesapeake Bay Retriever breed, in which the breed is listed as Chesa Bay Retr within the raw data. What this means is that pattern matching was needed to return a proper list of animals. Within this project regular expressions were utilized for this pattern matching. Pymongo does not accept regular expressions natively, and needed to be formatted differently in order for the Pymongo API to accept them.

## Contact

Your name: Oscar Rosa, Created for SNHU CS-340 on August 17, 2024

# Appendix

# Dash callbacks

#############################################

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)*

*])*

*])*

*])*

*]*