# CS 340 README Animal Shelter

## About the Project

This project was completed to fulfill the needs/requirements of Grazioso Salvare , an international rescue-animal training company.

## Motivation

This project exists to allow Grazioso Salvare to interface with Austin Animal Shelter to search for dogs that can fulfill their needs in the rescue-animal training arena. Grazioso Salvare identified specific dog breeds, sex and age requirements which was integrated into the website that has been created.

## Tools Used

1. MongoDB: <https://www.mongodb.com/>
   1. Rationale: MongoDB was chosen for it’s ease of use and robustness. It is known for its flexibility and scalability. MongoDB provide seamless integration with Python, which in turn allows for efficient data storage and retrieval.
2. Dash Framework: <https://dash.plotly.com/>
   1. Rationale: Dash framework is a user friendly framework for building web applications with Python. Utilizing Dash allows for easily implemented interactive dashboards.

## Functionality

Overall View Of Site:

A screenshot of a computer

Description automatically generated

Zoomed in view of Table:

A close-up of a document

Description automatically generated

Selections to filter per rescue type:



**Examples with each selection:**

Water Rescue:

A screenshot of a map

Description automatically generated

Mountain or Wilderness Rescue:

A screenshot of a computer

Description automatically generated

Disaster Rescue or Individual Tracking:

A screenshot of a map

Description automatically generated

## Project Recreation:

## Installation

The user will need the following tools to utilize this project. Links as to where these tools can be downloaded and installed from will accompany each tool:

* Python IDE
  + [Python IDE Wiki](https://wiki.python.org/moin/IntegratedDevelopmentEnvironments)
* Jupyter Notebook
  + [Jupyter Homepage](https://jupyter.org/)
* MongoDB
  + [Mongodb Homepage](https://www.mongodb.com/)
* Ploty Dash
  + [Ploty Dash Homepage](https://plotly.com/dash/)
* Dash Leaflet
  + [Dash Leaflet Homepage](https://www.dash-leaflet.com/)

## Project Recreation:

## Install components listed above.

1. Import Dataset

A screenshot of a computer

Description automatically generated

1. Add CRUD class and dashboard driver to server.
2. Update dashboard driver with username and password.

A screen shot of a computer

Description automatically generated

1. Add Dash components and their callbacks (Callbacks listed at the end of the document)
   1. Data table

A computer screen shot of a code

Description automatically generated

* 1. RadioItems

A screen shot of a computer code

Description automatically generated

* 1. Widgets

A screen shot of a computer code

Description automatically generated

## Challenges:

The main challenge that I faced with this project was getting my Imports correct. For the previous weeks project, I was unable to get my table and map to show properly at the same time. I struggled with this for a little over a week and finally requested help from my professor and he was able to point me in the right direction.

## Callbacks:

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

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

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

def update\_dashboard(filter\_type):

## FIX ME Add code to filter interactive data table with MongoDB queries

#button\_id = dash.callback\_context.triggered[0]['prop\_id'].split('.')[0]

if filter\_type == 'WR':

filtered\_cursor = shelter.read({'$and' : [{'sex\_upon\_outcome' : 'Intact Female'},

{'$or' : [

{'breed' : 'Labrador Retriever Mix'},

{'breed' : 'Chesa Bay Ret Mix'},

{'breed' : 'Newfoundland Mix'},

{'breed' : 'Newfoundland/Labrador Retriever'},

{'breed' : 'Newfoundland/Austrailian Cattle Dog'},

{'breed' : 'Newfoundland/Greate Pyenees'}]

},

{'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 26}},

{'age\_upon\_outcome\_in\_weeks': {'$lte': 156}}]

}]

})

Filtered\_data = list(filtered\_cursor)

#print("Filtered Data:", filtered\_data)

df\_filtered = pd.DataFrame(Filtered\_data)

elif filter\_type == 'MOWR':

filtered\_cursor = shelter.read({'$and' : [{'sex\_upon\_outcome' : 'Intact Male'},

{'$or' : [

{'breed' : 'German Shepherd'},

{'breed' : 'Alaskan Malamute'},

{'breed' : 'Old English Sheepdog'},

{'breed' : 'Siberian Husky'},

{'breed' : 'Rottweiler'}]

},

{'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 26}},

{'age\_upon\_outcome\_in\_weeks': {'$lte': 156}}]

}]

})

Filtered\_data = list(filtered\_cursor)

#print("Filtered Data:", filtered\_data)

df\_filtered = pd.DataFrame(Filtered\_data)

elif filter\_type == 'DROIT':

filtered\_cursor = shelter.read({'$and' : [{'sex\_upon\_outcome' : 'Intact Male'},

{'$or' : [

{'breed' : 'Doberman Pinscher'},

{'breed' : 'German Shepherd'},

{'breed' : 'Golden Retriever'},

{'breed' : 'Bloodhound'},

{'breed' : 'Rottwiler'}]

},

{'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 20}},

{'age\_upon\_outcome\_in\_weeks': {'$lte': 300}}]

}]

})

Filtered\_data = list(filtered\_cursor)

#print("Filtered Data:", filtered\_data)

df\_filtered = pd.DataFrame(Filtered\_data)

else:

df\_filtered = df.copy()

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

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

return data,columns

# Display the breeds of animal based on quantity represented in

# the data table

@app.callback(

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

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

def update\_graphs(derived\_virtual\_data):

###FIX ME ####

# add code for chart of your choice (e.g. pie chart) #

if derived\_virtual\_data is None:

return "No data available"

dff = pd.DataFrame.from\_dict(derived\_virtual\_data)

dff['percentage'] = dff['breed'].map(dff['breed'].value\_counts(normalize=True))

dff\_filtered = dff[dff['percentage'] >= 0.01]

fig = px.pie(

dff\_filtered,

names = 'breed',

values='percentage',

)

fig.update\_layout(

autosize=False,

margin=dict(l=20, r=20, t=80, b=20),

height=800,

width=800

)

return dcc.Graph(figure=fig)

# return [

# dcc.Graph(

# figure = px.pie(df, names='breed', title='Preferred Animals')

# )

# ]

#This callback will highlight a cell on the data table when the user selects it

@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]

# This callback will update the geo-location chart for the selected data entry

# derived\_virtual\_data will be the set of data available from the datatable in the form of

# a dictionary.

# derived\_virtual\_selected\_rows will be the selected row(s) in the table in the form of

# a list. For this application, we are only permitting single row selection so there is only

# one value in the list.

# The iloc method allows for a row, column notation to pull data from the datatable

@app.callback(

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

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

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

def update\_map(viewData, index):

if viewData is None:

return

elif index is None:

return

dff = pd.DataFrame.from\_dict(viewData)

# Because we only allow single row selection, the list can be converted to a row index here

if not dff.empty:

latitude = dff.iloc[index[0], 13]

longitude = dff.iloc[index[0], 14]

center = [latitude, longitude]

map\_children = [

dl.Map(

style = {'width': '800px', 'height': '800px'},

center = center,

zoom = 10,

children = [

dl.TileLayer(id="base-layer-id"),

dl.Marker(position = center, children=[

dl.Tooltip(dff.iloc[index[0], 4]),

dl.Popup([

html.H1("Animal Name"),

html.P(dff.iloc[index[0], 9])

])

])

])

]

return map\_children

## Contact

Your name: Christopher King