**CS-340 README For Grazioso Salvare Project**

**About the Project:**

The reason for creating this project was to build a functional web-based dashboard for the company Grazioso Salvare so that they have the capabilities to look at the Austin Animal Shelter Animals they have.

**Layers:**

This project consists of three layers: MongoDB, Python, a Plotly, Dash and Leaflet.

* MongoDB: This layer contains the data from an imported csv file from the animal shelter.
* Python: This is the “middle” level, this is where the code is implemented to create and control the third layer.
* Ploty, Dash and Leaflet: This is the third and final level of this project, this layer is what gives us the actual visualization of the dashboard all from the previous two layers.

At the python level, I have created a CRUD(Create, Read, Update, Delete) python project that does all the functions of the actual dashboard. This is all done in the MongoDB with the help of the Pymongo driver. This is where all the getter and setter methods are needed for the CRUD and MongoDB operations.

**Why MongoDB:**

The reason that we used MongoDB over something like SQL is due to how using mongo is much simpler and easy to navigate as a developer. Since MongoDB uses dynamic schema, this helps prevent errors during updating and creating operations.

**Why Python:**

Since python offers many tools for frontend and backend development, it seemed like the best choice. For example as mentioned before this project used the Pymongo Driver with allows for interaction between MongoDB and the CRUD python project which are both the back bone of the whole dashboard project. Using the Ploty-Dash and Dash Leaflet Libraries allowed me to use python to create HTML and or JavaScript based dashboards along with widgets.

**Why Ploty-Dash and Dash Leaflet Libraries:**

Plotly-Dash and Dash Leaflet are tools that use JavaScript but work with Python through their own APIs. They let you create things like SVG pie charts, tiled maps in PNG format, and the basic HTML and CSS to show things like data tables and widgets.

**The CRUD Class and Operations:**

CRUD(Create, Read, Update, Delete) are very common for wanting to create a database that allows queries. The only downside that I experienced during this project is the errors and handling said errors was monotonous. To make things simple getter and setter methods were implements and they are as follows:

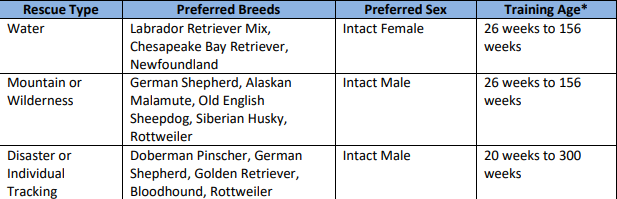
* The Constructor for username and password
* createRecord for data
* deleteRecord for queries
* getRecordId for data
* getRecordCriteria for criteria = none
* updateRecord for query and a newValue

**Project Requirements:**

The client Grazioso Salvare had the following requests for the dashboard:

* Logo provided by client
* Link to clients website
* A data table that allows custom filters and the Austin Animals Shelter information containing all of their animals
* Custom Filters requirements
  + Water Rescue
  + Mountain and Wilderness Rescue
  + Disaster Rescue
* Pie chart containing all breeds available
* A map showing the location of the animal queried

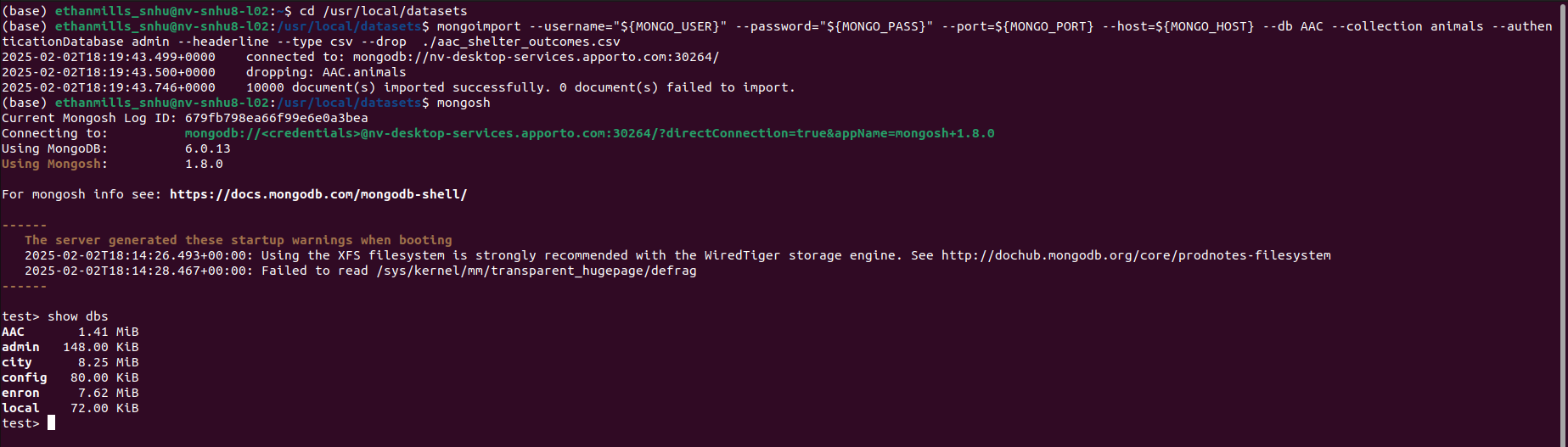
**Client Filter Requirements Visual:**

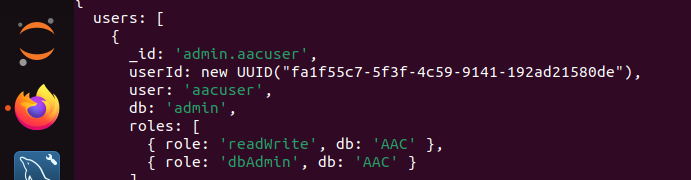


**Recreating Project:**

If for whatever reason this project needs to be recreated due to failed back up or any other reason, below are the steps to create this project step by step.

**MongoDB portion:**

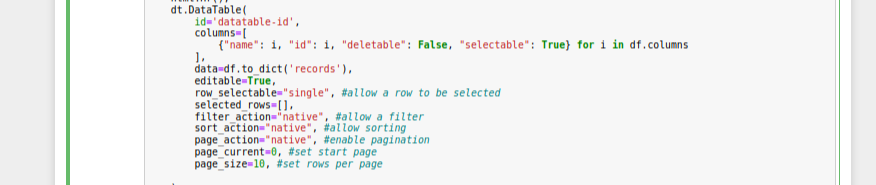
* Use an admin account to be able to load data into the MongoDB. Suggestion is to use Mongoimport
* Create an AAC user account name it whatever you'd like, example used in this project was:
  + Username: aacuser
  + Password: Password

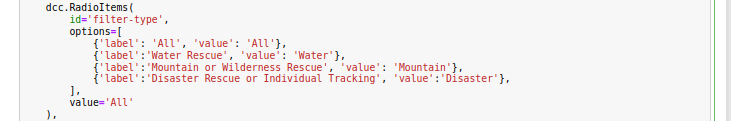


* Create CRUD python project:
  + Add crud class to project



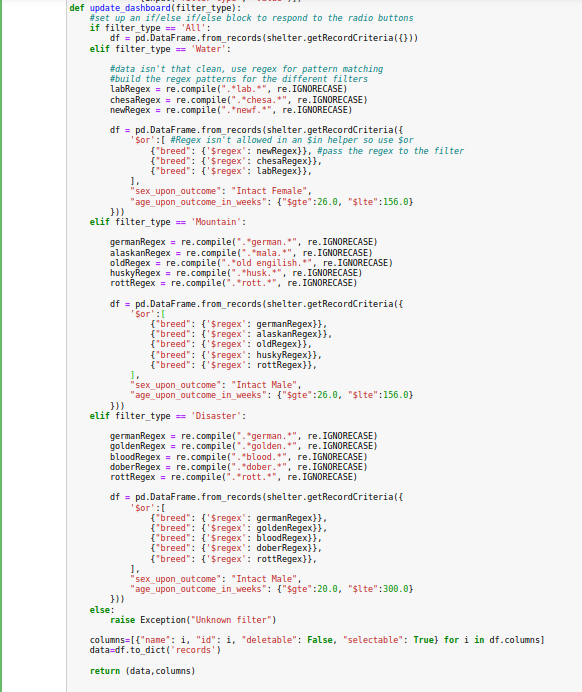
* Add Dash components
  + Data table



* HTLM widgets
* Radio Button Filters
* HTML anchor for clients logo 



* Filtering for Radio buttons



**My Struggles:**

My struggles with this project were far and wide. The main struggle I had was errors. Errors from within the MongoDB one especially being when trying to execute this project was the mongo port not working along with the same problem not working in the previous installment of this project. I also struggled with syntax errors in the Mongo Terminal. But through trial and error and a lot of learning I did learn what to look for to prevent errors and how to better code my projects. This project opens doors to potential future things that I could do as a developer.

**Dashboard Code:**

**/////////////**

from jupyter\_plotly\_dash import JupyterDash

import dash

import dash\_leaflet as dl

import dash\_core\_components as dcc

import dash\_html\_components as html

import plotly.express as px

import dash\_table as dt

from dash.dependencies import Input, Output, State

import os

import numpy as np

import pandas as pd

from pymongo import MongoClient

from bson.json\_util import dumps

import re #needed for the regex pattern matching

import base64 #need for images

from CRUD import AnimalShelter

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

# Data Manipulation / Model

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

username = "aacuser"

password = "Password"

shelter = AnimalShelter(password, username)

#create the default dataframe

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

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

# Dashboard Layout / View

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

app = JupyterDash('SimpleExample')

#add the customers branding

image\_filename = 'Grazioso Salvare Logo.png'

encoded\_image = base64.b64encode(open(image\_filename, 'rb').read())

app.layout = html.Div([

#create an anchor for the image/logo

#make the image an href to the website, www.snhu.edu

#open the link in a new tab by setting a blank target

html.A([

html.Center(html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()),

height = 250, width = 251))], href = 'https://www.snhu.edu', target = "\_blank"),

html.Center(html.B(html.H1('Ethan Mills SNHU CS-340 Dashboard'))),

html.Hr(),

#create the radio buttons to act as a filter

#set the default on initial load to to 'All'

dcc.RadioItems(

id='filter-type',

options=[

{'label': 'All', 'value': 'All'},

{'label':'Water Rescue', 'value': 'Water'},

{'label':'Mountain or Wilderness Rescue', 'value': 'Mountain'},

{'label':'Disaster Rescue or Individual Tracking', 'value':'Disaster'},

],

value='All'

),

html.Hr(),

dt.DataTable(

id='datatable-id',

columns=[

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

],

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

editable=True,

row\_selectable="single", #allow a row to be selected

selected\_rows=[],

filter\_action="native", #allow a filter

sort\_action="native", #allow sorting

page\_action="native", #enable pagination

page\_current=0, #set start page

page\_size=10, #set rows per page

),

html.Br(),

html.Hr(),

#This sets up the dashboard so that your chart and your geolocation chart are side-by-side

html.Div(className='row',

style={'display' : 'flex', 'justify-content':'center'},

children=[

html.Div(

id='graph-id',

className='col s12 m6',

),

html.Div(

id='map-id',

className='col s12 m6',

)

])

])

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

# 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)

])

])

])

]

**///////////**

**CRUD.py Code:**

from pymongo import MongoClient

from bson.objectid import ObjectId

import urllib.parse

class AnimalShelter(object):

#property variables

records\_updated = 0 #keep a record of the records updated in an operation; CYA

records\_matched = 0 #keep a record of the records macthed in an operation; CYA

records\_deleted = 0 #keep a record of the records deleted in an operation; CYA

#constructor to init the mongodb

#to do: this should be a singleton

def \_\_init\_\_(self, \_password, \_username = 'aacuser'):

#URI must be percent escaped as per pymongo documentation

userName = urllib.parse.quote\_plus('aacuser')

password = urllib.parse.quote\_plus('Password')

self.client = MongoClient('mongodb://%s:%s@localhost:27017/?authSource=AAC' %(userName, password))

self.dataBase = self.client['AAC']

#Mehtod to create a record

#Input data formatted as per the Pymongo API

#Example: ({""name": "Rex", 'age\_upon\_outcome': '2 months'})

def createRecord(self, data):

if data:

\_insertValid = self.dataBase.animals.insert\_one(data)

#check the status of the inserted value

return True if \_insertValid.acknowledged else False

else:

raise Exception("No document to save. Data is empty.")

#todo implement the R

#get documents by the GUID

#This is more for a test but could be used after the createRecord

#Since the document returned by insert\_one contains the newly created \_id

def getRecordId(self, postId):

\_data = self.dataBase.find\_one({'\_id': ObjectId(postId)})

return \_data

#Get records with criteria

#All records are returned if criteria is None

#Default is None

#Example: ({""name": "Rex", 'age\_upon\_outcome': '2 months'})

#do not return the \_id

def getRecordCriteria(self, criteria):

if criteria:

\_data = self.dataBase.animals.find(criteria, {'\_id' : 0})

else:

\_data = self.dataBase.animals.find({}, {'\_id' : 0})

return \_data

#Update a record

def updateRecord(self, query, newValue):

if not query:

raise Exception("No search criteria is present.")

elif not newValue:

raise Exception("No update value is present.")

else:

\_updateValid = self.dataBase.animals.update\_many(query, {"$set": newValue})

self.records\_updated = \_updateValid.modified\_count

self.records\_matched = \_updateValid.matched\_count

return True if \_updateValid.modified\_count > 0 else False

#delete a record

def deleteRecord(self, query):

if not query:

raise Exception("No search criteria is present.")

else:

\_deleteValid = self.dataBase.animals.delete\_many(query)

self.records\_deleted = \_deleteValid.deleted\_count

return True if \_deleteValid.deleted\_count > 0 else False