# CS 340 README Template

## About the Project/Project Title

I am employed at Global Rain, a software engineering firm that focuses on creating unique software. My group has been tasked with working on a project for Grazioso Salvare, an inventive worldwide rescue and animal training firm. Grazioso Salvare is searching for software that can recognize and classify dogs that are up for adoption using the data that animal shelters currently possess. A database and a client-facing web application dashboard are part of the complete stack development that Global Rain has been contracted to construct for this project.

## Motivation

The goal of the project is to enable users to add, search, edit, delete, and locate animals in the database. Categorization will also be an important aspect in achieving better search results. A client-side search tool was also implemented in order to better view the results via a data table, graph, and a map showing the location of the animal found, enabling better interpretation of results.

## Getting Started

Utilize the mongoimport tool via Terminal to import the AAC database (aac\_shelter\_outcomes.csv). Check your port value, as it can change from user to user. Enter the MongoShell and add a user (aacuser) specifically for the AAC database and Animal collection through the admin database. Ensure aacuser has read and write permissions for AAC database. Exit the MongoShell and test aacuser access by entering in the command “cd /usr/local/datasets/” followed by “MONGO\_USER=aacuser” and “MONGO\_PASS=<password>” where <password> is the password you set when creating the user. Reenter the MongoShell, if it goes through successfully the user has been authenticated correctly.

The client portal has multiple functions to assist in filtering specific aspects of the data from the database. There are specific rescue types, “Water Rescue”, “Mountain/Wilderness Rescue”, “Disaster Rescue/Individual Tracking”, and an option to Reset back to the full database result. Users will also have access to changing the data showed on the pie graph based on Breed, Age, and Outcome Type, along with a map to show the latitude and longitude of the animal.

## Installation

Tools used are via Linux.

Python (latest version recommended) needs to be installed on the system.

Jupyter Notebook for editing and running Python code.

MongoDB is required as the library imports for Python are focused on MongoDB.

Library Imports used:

“from pymongo import MongoClient”

“from bson.objectid import ObjectId”

Creating the client facing web portal to view the data required some additional tools for installation:

“from jupyter\_dash import JupyterDash”

“import dash\_leaflet as dl”

“from dash import dcc”

“from dash import html”

“import plotly.express as px”

“from dash import dash\_table”

“from dash.dependencies import Input, Output, State”

“import base64”

“import os”

“import numpy as np”

“import pandas as pd”

“import matplotlib.pyplot as plt”

“from pymongo import MongoClient”

“from bson.json\_util import dumps”

“import re”

## Usage

### Code Example

The library in its current state allows for the addition and search for animals within the AAC Animal database. By utilizing def create(self, data):

if data is not None:

self.database.animals.insert\_one(data) # data should be dictionary

return True

else:

#Exception

raise Exception("Nothing to save, because data parameter is empty")

return False

and def read\_all(self, data):

cursor = self.database.animals.find(data, {"\_id" :False}) #Return cursor which points to documents

return cursor

def read(self, data):

return self.database.animals.find(data) #Return document as dictionary

where data is the information concerning the animal you wish to add or find.

Additions have recently been made for updating and deleting entries to the database. Updating entries are processed through:

def update(self, query, update\_data):

try:

result = self.collection.update\_many(query, {"$set": update\_data})

return result.modified\_count

except Exception as e:

# Handle the exception here, you can log the error message or raise a custom exception

print(f"An error occurred during update: {e}")

return 0 # Return 0 indicating no documents were modified

where query is the animal information being searched for to be updated, and where update\_data is the new data being updated into the animal entry.

Deleting entries are processed via:

def delete(self, query):

try:

result = self.collection.delete\_many(query)

return result.deleted\_count

except Exception as e:

# Handle the exception here, you can log the error message or raise a custom exception

print(f"An error occurred during delete: {e}")

return 0 # Return 0 indicating no documents were deleted

where query is the information used to find the animal to be removed from the database.

### Tests

Tests are done with the .ipynb file to import the animal\_shelter.py file. The data variable is then edited to add, find, update, or delete the animal(s) in question. An example of adding an animal could look like this:

data = {"age\_upon\_outcome" : "2 years",

"animal\_id" : "Test",

"animal\_type" : "Cat",

"breed" : "American DSH",

"color" : "Tuxedo",

"date\_of\_birth" : "2016-11-15",

"datetime" : "2016-11-15 17:00:00",

"monthyear" : "2016-11-15T17:00:00",

"name" : "Bagheera",

"outcome\_suptype" : "",

"outcome\_type" : "Adoption",

"sex\_upon\_outcome" : "Neutered Male",

"location\_lat" : 30.6737365854231,

"location\_long" : -97.707971529467,

"age\_upon\_outcome\_in\_weeks" : 104.285714}

if shelter.create(data):

print("Animal has been added")

Finding an animal could look like this:

query = shelter.read({"name" : "Bagheera"})

for animal in query:

print(animal)

Updating animal information currently was achieved through:

updateAnimal = shelter.update({"name" : "Bagheera"}, {"outcome\_type" : "Lived happily ever after."})

print(updateAnimal , " entries updated.")

Deleting animals from the database was processed via:

updateAnimal = shelter.update({"name" : "Bagheera"}, {"outcome\_type" : "Lived happily ever after."})

print(updateAnimal , " entries updated.")

**Client Portal**

Some examples that provide utility to the client side portal are as follows:

Specialized data sort requested by the client:

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

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

Output('datatable-id','selected\_rows')],

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

def update\_dashboard(filter\_type):

# Code to filter interactive data table with MongoDB queries

# Query database for characteristics specified by client for each filter type

# Additionally, exclude euthanized animals

if filter\_type == 'RESET':

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

elif filter\_type == 'WR':

#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(list(shelter.readAll({

"animal\_type": "Dog",

'$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": {"$gte": 26, "$lte": 156},

"outcome\_type": {"$nin": ["Euthanasia"]}

})))

elif filter\_type == 'MWR':

df = pd.DataFrame(list(shelter.readAll({

"animal\_type":"Dog",

"breed":{"$in":["German Shepherd", "Alaskan Malamute", "Old English Sheepdog",

"Siberian Husky", "Rottweiler"]},

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

"age\_upon\_outcome\_in\_weeks":{"$gte": 26},

"age\_upon\_outcome\_in\_weeks":{"$lte": 156},

"outcome\_type":{"$nin":["Euthanasia"]}

})))

elif filter\_type == 'DRIT':

df = pd.DataFrame(list(shelter.readAll({

"animal\_type":"Dog",

"breed":{"$in":["Doberman Pinscher", "German Shepherd", "Golden Retriever",

"Bloodhound", "Rottweiler"]},

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

"age\_upon\_outcome\_in\_weeks":{"$gte": 20},

"age\_upon\_outcome\_in\_weeks":{"$lte": 300},

"outcome\_type":{"$nin":["Euthanasia"]}

})))

else:

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

# Prepare data and columns to be sent to data table

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

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

# Reset selected row to first row when filter setting changes

selected\_rows = [0]

return (data, columns, selected\_rows)

The code as it sits right now doesn’t display the Water Rescue Breeds in its entirety, I tried to use the regex utility so that it would show any and all variations but only had success in showing the Labrador breed and its mixes.

Chart data is shown via:

@app.callback(

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

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

Input('pie\_dropdown', 'value')])

def update\_graphs(viewData, dropdownValue):

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

return [

dcc.Graph(

figure = px.pie(

data\_frame=dff,

names=dropdownValue)

)

]

And the map data is shown via:

@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 or not index:

return []

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

if dff.empty:

return []

row = index[0] if index else 0

if row >= len(dff):

return []

return [

dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.75,-97.48], zoom=10, children=[

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

dl.Marker(position=[dff.iloc[row,13],dff.iloc[row,14]], children=[

dl.Tooltip(dff.iloc[row,4]),

dl.Popup([

html.H1("Animal Name"),

html.P(dff.iloc[row,9])

])

])

])

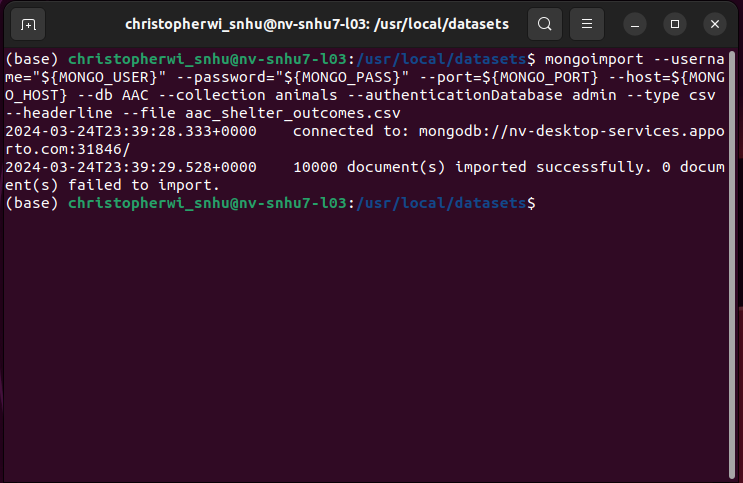
]

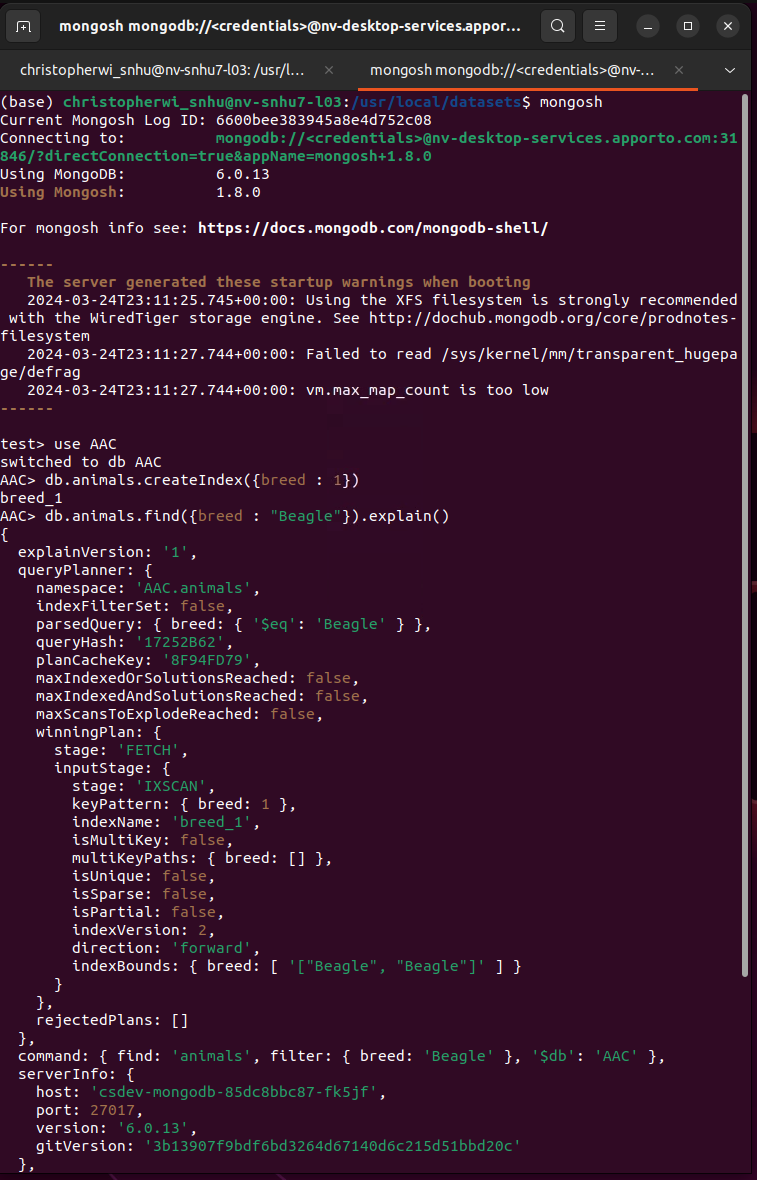
These are snippets of the code, the full code can be view via the screenshots below.

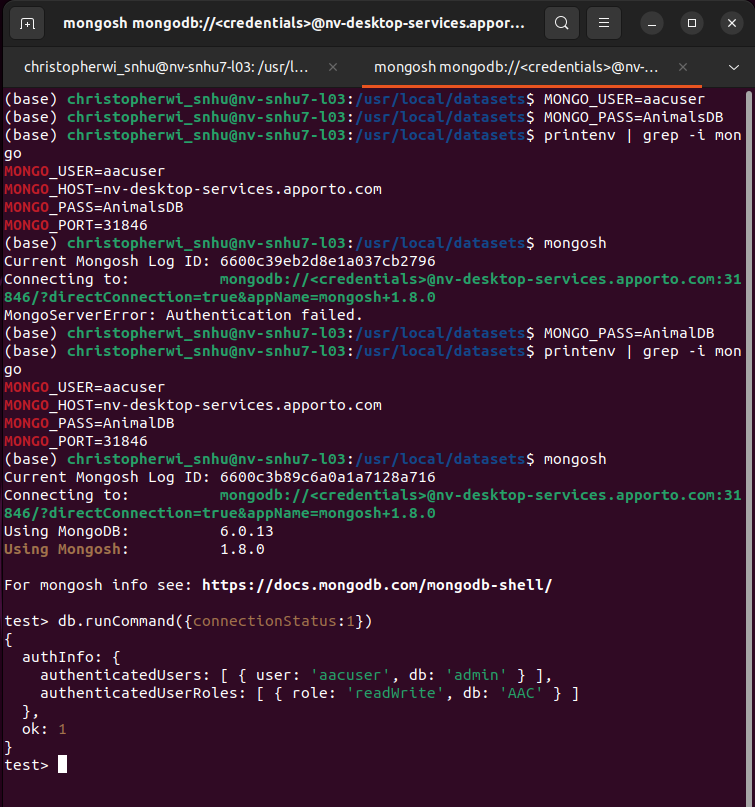
### Screenshots

Screenshots of the code mentioned above are as follows:

**MongoDB - Terminal**







**animal\_shelter.py**

A screenshot of a computer program

Description automatically generated

Tests done in order of create, read, update, and delete:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Client Portal Code**

A screenshot of a computer program

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated

**Client Portal Experience**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

## Contact

Your name: Chris Wills