# CS 340 PROJECT: README

## About the Project/Project Title

Canine Recruitment

This project is to help the client, Grazioso Salvare, identify dogs that are good candidates for Search and Rescue Training.

## Motivation

The search and Rescue Training program aims to train canines to locate and rescue humans or other animals in threatening conditions. Dogs under two years old usually undergo search and rescue training more successfully. Moreover, certain breeds are more adept at specific types of rescues. Therefore, this application allows the user to access the database to **create, read, update, and delete** the existing data from the animal shelters. These functions can help users manage persistent data, such as adding new and modifying documents, searching and retrieving specific criteria from the collection, and removing unnecessary database records.

## Steps to complete the project.

1. Access MongoDB using the administrator account to import the local data (CSV file) into the database.
2. Create simple and compound indexes to improve performance. It can avoid the need to scan every document in a collection that matches the query.
3. Create a user account to provide the security needs and ensure user authentication to the database.
4. Create a Python module and enable functions. Then, use the Python module to develop a web application that connects the client-side user interface, the dashboard to the database.
5. Review the specification document.
6. Create a dash app with Python, import necessary libraries and CRUD module.
7. Build interactive visualization and dashboard.
8. Add Widgets.
   1. Interactive data table
   2. Filter buttons
   3. Pie chart
   4. Geolocation graph
9. Testing the dashboard and verifying functionally.
10. Construct a ReadMe file to document the project.

## Installation

#### MongoDB to access the database, import the Austin Animal Center (aac) CSV file, and create a user account.

* + The insert command can insert one or many documents and return to true or false.
  + The find command returns documents, returning a cursor to the documents.
  + The update/updateMany methods can modify specific fields of an existing document.
  + The delete/deleteMany command removes documents and returns the value of documents deleted.
  + createUser to create credentials externally to MongoDB. It would return an error if the user already existed on the database.

#### Jupyter Notebook: create the Python module to enable functions.

* + Connect MongoDB with Python using PyMongo.

from pymongo import MongoClient

self.client = MongoClient(f'mongodb://{user}:{password}@{HOST}:{PORT}/{DB}?authSource={AUTH\_DB}')

self.database = self.client[DB]

self.collection = self.database[COL]

#### Dash Framework provides view and controller structure for the web application.

For this project, we used Dash, an open-source framework provided by Plotly, to build data visualization interfaces for our client.

Building the layout using dash html components and dash core components

* + - * + from dash import dcc

dcc.Graph

dcc.Geolocation

* + - * + from dash import html

html.H1

html.Button

## Usage

### Mongo Code Example

#### user authentication to the database and collection

cd /usr/local/datasets/

MONGO\_USER=aacuser

MONGO\_PASS=SNHU1234

mongoimport --username=${MONGO\_USER} \

--password=${MONGO\_PASS} --port=${MONGO\_PORT} \

--host=${MONGO\_HOST} --db AAC --collection animals \

--authenticationDatabase AAC --type csv --headerline --file aac\_shelter\_outcomes.csv

#### Tests

* Inserting a CSV file using the appropriate MongoDB import tool.
  + Return -> 10000 documents imported successfully.
* User authentication to the database and collection
  + Login processed

#### Screenshots

A screenshot of a computer program

Description automatically generated

### CURD Code Example

#### # Create method to implement the C in CRUD

def create(self, data):

if data is not None:

successfulInsert = self.collection.insert(data)

if successfulInsert is not None:

return True

return False

else:

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

#### Tests

A Create method that inserts a document into a specified MongoDB database and collection.

* Input -> argument to function should be the key/value lookup pair to use with the MongoDB driver find API call.
* Return -> "True" if successful insert, else "False".

#### Screenshots

For this scenario, I inserted a tested data. Returned True.A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

#### # Read method to implement the R in CRUD

def read(self, \*args): # \*args to pass a number of arguments

query = {} #empty dictionary to combine search criteria

for arg in args:

query.update(arg)

found = False

print("Searching Data")

for animals in self.collection.find(query):

print(animals)

found = True

if not found:

print("No such data")

#### Tests

* Input -> arguments to function should be the key/value lookup pair to use with the MongoDB driver find API call.
* Return -> result in a list if the command is successful, else an empty list.

#### Screenshots

Users can enter search criteria to find the animals that match the data. For this scenario, as mentioned, dogs under 2 years old are generally more effective for search and rescue training. Therefore, I searched dogs under 2 years old, less than 104.286 weeks. I also tested the empty list.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

#### # Update Method to implement the R in CRUD

def update(self, \*args, \*\*kwargs):

findExistData = {} # Empty dictionary to combine search criteria

setUpdate = {}

for search in args:

findExistData.update(search)

for newData in kwargs.items():

setUpdate.update({newData[0]: newData[1]})

if findExistData is not None:

updated\_result = self.collection.update\_many(findExistData, {"$set": setUpdate})

modified\_count = updated\_result.raw\_result.get('nModified', 0)

print(f"Number of modified documents: {modified\_count}")

return updated\_result.raw\_result

else:

raise Exception("Nothing to update because the search criteria is empty")

#### Tests

* Input -> arguments to function should be the key/value lookup pair to use with the MongoDB driver Find API call.
* return -> The number of objects modified in the collection.

#### Screenshots

For this scenario, I modified age\_upon\_outcome and animal\_type under the name: TEST and animal\_id: test. It returned a number of modified documents and also I called the read function to confirmed the update. A screenshot of a computer

Description automatically generated

#### # Delete Method to implement the D in CRUD

def delete(self, removeData):

if removeData is not None:

deleteData = self.collection.delete\_many(removeData)

deleted\_count = deleteData.raw\_result.get('n', 0) # Use 'n' instead of 'nModified'

print(f"Number of deleted documents: {deleted\_count}")

return deleteData.raw\_result

else:

raise Exception("Nothing to delete because the search criteria is empty")

#### Tests

* Input -> arguments to function should be the key/value lookup pair to use with the MongoDB driver find API call.
* Return -> The number of objects removed from the collection.

#### Screenshots

For this scenario, I removed the documents with the name: TEST and animal\_id: test. It returned a number of deleted documents and also I called the read function to confirmed the removal.

A screenshot of a phone

Description automatically generated

### Web Application Dashboard (Python Code)

#### Configure the necessary Python module imports

# Setup the Jupyter version of Dash

from jupyter\_dash import JupyterDash

# Configure the necessary Python module imports

import dash\_leaflet as dl

from dash import Dash

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

# Configure OS routines

import os

# change animal\_shelter and AnimalShelter to match your CRUD Python module file name and class name

from AnimalShelters import AnimalShelter

# Configure the plotting routines

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#### # Data Manipulation / Model

# username and password and CRUD Python module name.

username = "aacuser"

password = "SNHU1234"

db = AnimalShelter(username, password)

df = pd.DataFrame.from\_records(db.read({}))

df.drop(columns=['\_id'], inplace=True)

#### # Dashboard Layout / View

app = JupyterDash(\_\_name\_\_)

app.previous\_clicks = {'button0': 0, 'button1': 0, 'button2': 0, 'button3': 0}

app.layout = html.Div([

html.Div(id='hidden-div', style={'display':'none'}),

html.Center(html.B(html.H1('SNHU CS-340, KaLee Li Dashboard'))),

html.Hr(),

html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()),style={'height':'20%','width':'20%'}),

html.Div(className='buttonRow',

style={'display' : 'flex'},

children=[

html.Button(id='submit-button-zero', n\_clicks=0, children='Reset To All', style={'fontSize': '20px'}),

html.Button(id='submit-button-one', n\_clicks=0, children='Water', style={'fontSize': '20px'}),

html.Button(id='submit-button-two', n\_clicks=0, children='Mountain/Wilderness', style={'fontSize': '20px'}),

html.Button(id='submit-button-three', n\_clicks=0, children='Disaster/Individual Tracking', style={'fontSize': '20px'})

]),

dash\_table.DataTable(

id='datatable-id',

columns=[

{"name": i, "id": i, "deletable": False, "selectable": True}

for i in df.columns

],

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

editable=False,

filter\_action="native",

sort\_action="native",

sort\_mode="multi",

column\_selectable=False,

row\_selectable="single",

row\_deletable=False,

selected\_columns=[],

selected\_rows=[0],

page\_action="native",

page\_current=0,

page\_size=10

),

html.Br(),

html.Hr(),

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

html.Div(className='row',

style={'display' : 'flex'},

children=[

html.Div(

id='graph-id',

className='col s12 m6',

),

html.Div(

id='map-id',

className='col s12 m6',

)

])

],style={'textAlign': 'center'})

#### # Interaction Between Components / Controller

##### *Callback: highlight a row 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]

##### *Callback: update the geo-location chart for the selected data entry*

@app.callback(

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

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

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

)

def update\_map(viewData, index):

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

if dff.empty:

# Handle the case when the DataFrame is empty

return []

if index is None or len(index) == 0:

row = 0

else:

row = index[0]

# Austin TX is at [30.75,-97.48]

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])

])

])

])

]

##### *Callback: update the pie chart breeds based on quantity represented in the data table*

@app.callback(

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

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

def update\_graphs(viewData):

###FIX ME ####

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

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

# Check if the DataFrame is empty

if dff.empty:

return []

return [

dcc.Graph(

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

)

]

##### *Callback: filter breed and age based on training expertise.*

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

[Input('submit-button-zero', 'n\_clicks'),

Input('submit-button-one', 'n\_clicks'),

Input('submit-button-two', 'n\_clicks'),

Input('submit-button-three', 'n\_clicks')

])

def on\_click(button0, button1, button2, button3):

# start case

df = pd.DataFrame.from\_records(db.read({}))

#Filter Logic

if button0 > app.previous\_clicks['button0']:

df = pd.DataFrame.from\_records(db.read({}))

elif button1 > app.previous\_clicks['button1']:

df = pd.DataFrame.from\_records(db.read({

"breed": {

"$in": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]},

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

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

}))

elif button2 > app.previous\_clicks['button2']:

df = pd.DataFrame.from\_records(db.read({

"breed": {

"$in": ["German Shepherd",

"Alaskan Malamute",

"Old English Sheepdog",

"Siberian Husky",

"Rottweiler"]},

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

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

}))

elif button3 > app.previous\_clicks['button3']:

df = pd.DataFrame.from\_records(db.read({

"breed": {

"$in": ["Doberman Pinscher",

"German Shepherd",

"Golden Retriever",

"Bloodhound",

"Rottweiler"]},

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

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

}))

# Cleanup Mongo \_id field

df.drop(columns=['\_id'],inplace=True)

# Update previous click counts

app.previous\_clicks['button0'] = button0

app.previous\_clicks['button1'] = button1

app.previous\_clicks['button2'] = button2

app.previous\_clicks['button3'] = button3

return df.to\_dict('records')

### Dashboard Functionality Screenshots

#### The Grazioso Salvare logo and unique identifier.

A screenshot of a calendar

Description automatically generated­­

#### Interactive data table, the Pie chart, and a Geolocation Chart

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#### Filter Options

##### *Water Rescue*

A screenshot of a computer

Description automatically generated

##### *Mountain or Wilderness Rescue*

A screenshot of a computer

Description automatically generated

##### *Disaster Rescue or Individual Tracking*

A screenshot of a computer screen

Description automatically generated

##### *Reset (returns all widgets to their original, unfiltered state)*

A screenshot of a computer

Description automatically generated

## Challenges

The biggest struggle I encountered working on this project was authentication failure to log into the user account I created. I tried troubleshooting by changing the user password, dropping a user, and re-creating the user account, but it was still unsuccessful. Hence, I contacted our professor, along with the error screenshots, and explained the methods I've tried to debug. The professor offered suggestions, and the problem was solved using:

* mongosh --authenticationDatabase AAC
* MongoClient(f'mongodb://{user}:{password}@{HOST}:{PORT}/{DB}?authSource={AUTH\_DB}')

## Reference Links

*DAsh Core Components | DAsh for Python Documentation | Plotly*. (n.d.). <https://dash.plotly.com/dash-core-components>

*DAsh DataTable | DAsh for Python Documentation | Plotly*. (n.d.). <https://dash.plotly.com/datatable>

*DASH Framework*. (n.d.). <https://www.tutorialspoint.com/python_web_development_libraries/python_web_development_libraries_dash_framework.htm>

*Dash HTML Components | Dash for Python Documentation | Plotly*. (n.d.). <https://dash.plotly.com/dash-html-components>

*Pie*. (n.d.). <https://plotly.com/python/pie-charts/>

## Contact

name: KaLee Li