# CS 340 README Project Two

## About the Project/Project Title

The objective of this project is to create a dashboard for Grazioso Salvare through my position at Global Rain. The main goal was to implement a dashboard that utilized the methods of CRUD (Create, Read, Update, and Delete) to help gather and search for information based on the animals in the Austin Animal Center shelter home and respective locations. Side objectives of this project included interactive UI features such as radio buttons to search for animals in specific areas, and a drop-down menu for different labelling on the included pie chart (i.e. Breed, Outcome, etc.)

## Motivation

The motivation behind this project is to test my skills and knowledge in CRUD operations and return visual data stored within databases.

## Getting Started

To get started locally:

1. Ensure installation of MongoDB on your working device.
2. Create a database within MongoDB with a simple yet effective and informative name (i.e. AAC, for Austin Animal Control).
3. Create a user with permission to read and write within the database.  
   A screenshot of a computer program

   Description automatically generated
4. Import the necessary csv file using mongoimport to insert known data into the database.  
   A screenshot of a computer

   Description automatically generated
5. Launch your version of the mongo shell (i.e. mongosh) and perform the proper action (use <DB NAME HERE>)
6. Install/access your python IDE/Notebook and run the program used to perform the steps of CRUD (Create, Read, Update, Delete).
7. By instantiating the created CRUD python module into the ProjectTwoDashboard.ipynb file the dashboard can be accessed via the port connected to your MongoDB, in my case the port was 30968 (port numbers WILL vary).

## Installation

The following installations are required:

1. Most recent/current version of MongoDB
2. Most recent/current version of Python and IDE of choice
3. Most recent/current version of Jupyter Notebook

## Usage

### Code Example

The following is an example of the Python code used to run the create and read functions as well as the method for ensuring proper connection to MongoDB:

**The following is the CRUD python module (CRUD.py)**

from pymongo import MongoClient

from bson.objectid import ObjectId

class AnimalShelter:

    #CRUD operations for Animal collection in MongoDB

    #Initializing MongoDB Client

    def \_\_init\_\_(self, username, password):

        #Initialize connection

        self.client = MongoClient('mongodb://%s:%s@nv-desktop-services.apporto.com:30968' % (username, password))

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

    #Create method

    def create(self ,data):

        if data is not None:

            self.database.animals.insert\_one(data)

            return True

        else:

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

    #Read method

    def read(self, search):

        if search is not None:

            result = self.database.animals.find(search)

            for animal in result:

                print(animal)

        else:

            raise Exception("Nothing to find, search parameter is empty")

        return result

#Update method

def update(self, search, updateInfo):

numAnimalsUpdated = 0

if search and updateInfo is not None:

result = self.database.animals.find(search)

for animal in result:

self.database.animals.update(search, updateInfo)

numAnimalsUpdated += 1

else:

raise Exception("Nothing to be updated due to empty search parameter or unchanged information")

print("Number of entries updated: ", numAnimalsUpdated)

#Delete method

def delete(self, search):

numAnimalsDeleted = 0

if search is not None:

result = self.database.animals.find(search)

for animal in result:

self.database.animals.delete\_one(search)

numAnimalsDeleted += 1

else:

raise Exception("Nothing to delete due to empty search parameter")

print("Number of entries deleted: ", numAnimalsDeleted)

**The following is the code from the ProjectTwoDashboard.ipynb file**

# Setup the Jupyter version of Dash

from jupyter\_dash import JupyterDash

# Configure the necessary Python module imports for dashboard components

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

# Configure OS routines

import os

# Configure the plotting routines

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pymongo import MongoClient

from bson.json\_util import dumps

from CRUD import AnimalShelter

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

# Data Manipulation / Model

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

username = "aacuser"

password = "CmS1201"

# Connect to database via CRUD Module

db = AnimalShelter(username, password)

# class read method must support return of list object and accept projection json input

# sending the read method an empty document requests all documents be returned

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

## Debug

# print(len(df.to\_dict(orient='records')))

# print(df.columns)

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

# Dashboard Layout / View

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

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

# Add logo

image\_filename = 'Grazioso Salvare Logo.png' # replace with your own image

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

app.layout = html.Div([

# Creation of an anchor for the logo

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

# Open 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('Corey Sampsons\' CS-340 Dashboard'))),

html.Hr(),

# Creation of radio buttons to act as a filter

# Default set to "All"

html.Div(

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'},

]

)

),

html.Hr(),

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=True,

row\_selectable="single", # Allow for single selectable row

selected\_rows=[],

filter\_action="native", # Allow for filtering

sort\_action="native", # Allow for 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'},

children=[

html.Div([

dcc.Dropdown(

id='pie\_dropdown',

className='col s12 m6',

options=[

{'label': 'Breed', 'value': 'breed'},

{'label': 'Age', 'value': 'age\_upon\_outcome'},

{'label': 'Outcome', 'value': 'outcome\_type'}

],

value = 'breed',

multi=False,

clearable=False

)

],

style = {"width":"10%"}

),

html.Div(

id='graph-id',

className='col s12 m6',

style={"width":"45%"}

),

html.Div(

id='map-id',

className='col s12 m6',

style={"Width":"45%"}

)

])

])

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

# Interaction Between Components / Controller

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

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

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

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

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

def update\_dashboard(filter\_type):

if filter\_type == 'All':

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

elif filter\_type == 'Water':

df = pd.DataFrame(list(db.read({

"animal\_type":"Dog",

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

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

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

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

})))

elif filter\_type == 'Mountain':

df = pd.DataFrame(list(db.read({

"animal\_type":"Dog",

"breed":{"$in":["Germn 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}

})))

elif filter\_type == 'Disaster':

df = pd.DataFrame(list(db.read({

"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}

})))

else:

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

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

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

selected\_rows = [0]

return (data, columns, selected\_rows)

# Changes color of a selected cell

@app.callback(

Output('datatable-id', 'style\_data\_conditional'),

[Input('datatable-id', 'selected\_rows')]

)

def update\_styles(selectedRows):

return [{

'if': { 'row\_index': i },

'background\_color': '#D2F3FF'

} for i in selectedRows]

# Callback for pie chart

# Set to plot all of the data across all the pages instead of viewable data

@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)

)

]

# 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\_selected\_rows")])

def update\_map(virtualRows):

if not virtualRows:

Array = (30.75, -97.48)

toolTip = "Austin Animal Center (AAC)"

popUpHeading = "Austin Animal Shelter"

popUpParagraph = "Shelter Home Location"

else:

dff = pd.DataFrame(df.iloc[virtualRows])

coordLat = float(dff['location\_lat'].to\_string().split()[1])

coordLong = float(dff['location\_long'].to\_string().split()[1])

Array = (coordLat, coordLong)

toolTip = dff['breed']

popUpHeading = "Animal Name"

popUpParagraph = dff['name']

return [dl.Map(style={'width': '1000px', 'height': '500px'}, center=Array,

zoom=10, children=[dl.TileLayer(id="base-layer-id"),

dl.Marker(position=Array, children=[

dl.Tooltip(toolTip),

dl.Popup([

html.H1(popUpHeading),

html.P(popUpParagraph)

])

])

])

]

app.run\_server()

### Tests

The following are tests run to ensure proper functionality of connecting to the DB and valid use of the create and read methods:

1. Under the main function a variable called animalList calls to the AnimalShelter class in the CRUD.py file using the user credentials to login.
2. Data is then created for the creation method to ensure functionality (i.e. record number, animal type, name, etc.).
3. A search is performed using the search method to ensure proper read functionality.
4. If/else statement is run to inform the user whether the data inserted was successful or not.
5. A call to the read function that displays a list with corresponding tags.
6. A call to the update function which displays when an entry in the database has been updated
7. A call to the delete function which displays when an entry has been removed from the database

### Screenshots*A screenshot of a computer Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

**DASHBOARD OVERVIEW**

**A screenshot of a computer

Description automatically generated  
A screenshot of a map

Description automatically generated**

**DASHBOARD RADIO BUTTONS**

**A screenshot of a computer

Description automatically generated**

Figure 1 Disaster Rescue or Individual Tracking

**A screenshot of a computer

Description automatically generated**

Figure 2 Mountain or Wilderness Rescue

**A screenshot of a computer

Description automatically generated**

Figure 3 Water Rescue

**DASHBOARD PIE CHART AND PIE CHART DROPDOWN MENU**

**A screenshot of a computer screen

Description automatically generatedA screenshot of a map

Description automatically generated**

**DASHBOARD INTERACTIVE MAP (SELECTED RECORD NUMBER 17, Gigi)**

**A screenshot of a map

Description automatically generated**

**Obstacles**

While there were a lot of small trial and error methods and bug fixes to perform here and there while trying to get the dashboard to run properly, most of this felt as if it came naturally. The main issue I did run into was the initial connection of the database to the dashboard. This issue caused some headaches and frustrations to boil over at times during the process, however, I concluded that it was a simple typo in my python module that set everything awry. Because of this simple misstep, I plan to take extra time and caution when reviewing and re-reviewing my code from now on.

## Contact

Name: Corey Sampson

Email: corey.sampson@snhu.edu