# CS 340 README Project One

## About the Project

This project is for a company seeking a software application that identifies and categorizes dogs based on existing data from animal shelters. The database and client-facing web application also need to be implemented for this project. This project must also be open source on GitHub for other organizations to utilize.

## Motivation

This project is being created to aid in the identification of dogs that are capable of being trained to rescue humans or other animals. This project will be maintained by frequently updating the database so all dogs can be identified and determined if they are capable of being trained.

## Getting Started

To get a local copy up and running, you will need a distribution of Ubuntu Linux, as well as MongoDB, Python, and Jupyter Notebook.

1. Install the latest version of MongoDB on your Linux machine.
2. Open the Terminal and change the directory to where the database is located.
3. Once the directory has been changed, open MongoDB by entering “mongosh”.
4. Type “exit” to return to the Linux shell.
5. Import the database using *mongoimport*.
6. Once the database has been imported, use the code provided in the code example to implement creating, reading, updating, and deleting a document.

## Installation

Ubuntu, MongoDB, Python, PyMongo, Jupyter Notebook.

**Purpose of the CRUD Python Module**

The CRUD Python module is designed to facilitate basic database operations: Create, Read, Update, and Delete (CRUD) on a MongoDB database. It encapsulates these operations into methods that can be easily called from within a Python script, providing a convenient interface for interacting with MongoDB collections.

**How the Module Should Be Used**

1. Import the Module: Import the CRUD module into your Python script or application.
2. Instantiate the Class: Create an instance of the AnimalShelter class provided by the module.
3. Use CRUD Operations:

* Create: Call the create method to insert documents into the MongoDB collection.
* Read: Call the read method to query documents from the collection.
* Update: Call the update method to modify existing documents in the collection.
* Delete: Call the delete method to remove documents from the collection.

**Python Driver for MongoDB**

The Python driver pymongo is the official MongoDB driver for Python and is widely used in the Python community for interacting with MongoDB databases. It provides a large set of features and is actively maintained by MongoDB, Inc.

**Why MongoDB**

MongoDB was chosen for several reasons, including its flexibility, scalability, and ease of use with Python. MongoDB allows for flexible data storage, and the query language is powerful and expressive, allowing for complex queries to be performed on the data, which in this case, is necessary. Another benefit of MongoDB, is the ability to scale horizontally, meaning it can easily handle large amounts of data and high traffic loads.

**Dash Framework**

Dash is a Python framework for building web applications. It is built on top of Flask, Plotly.js, and React.js, and it allows for the creation of interactive web applications purely in Python. Also, Dash uses callback functions to update the content of your application in response to user input or other events, which is an important tool for the filter options in the dashboard.

**Attributes and Functionality of CRUD Operations**

* Create: Insert a document into the specified MongoDB database and collection.
* Read: Queries for documents from the specified MongoDB database and collection.
* Update: Updates document(s) in the specified MongoDB database and collection based on specified criteria.
* Delete: Removes document(s) from the specified MongoDB database and collection based on specified criteria.

### Code Example

**Python CRUD Module**

**from** pymongo **import** MongoClient

**from** bson**.***objectid* **import** ObjectId

**class** **AnimalShelter(object):**

""" CRUD operations for Animal collection in MongoDB """

**def** \_\_init\_\_**(**self**,** username**=**'aacuser'**,** password**=**'hockey4'**,** host**=**'nv-desktop-services.apporto.com'**,** port**=**32418**,** db**=**'AAC'**,** col**=**'animals'**):**

self**.***client* **=** MongoClient**(**'mongodb://%s:%s@%s:%d' **%** **(**username**,** password**,** host**,** port**))**

self**.***database* **=** self**.***client***[**db**]**

self**.***collection* **=** self**.***database***[**col**]**

**def** create**(**self**,** data**):**

"""Inserts a document into the specified MongoDB database and collection."""

**if** data**:**

**try:**

self**.***collection***.***insert\_one***(**data**)**

**return** **True**

**except** **Exception** **as** e**:**

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

**return** **False**

**else:**

**print(**"Nothing to save because data parameter is empty"**)**

**return** **False**

**def** read**(**self**,** filter\_query**=None):**

"""Queries for documents from the specified MongoDB database and collection."""

**try:**

**if** filter\_query **is** **None:**

result **=** self**.***collection***.***find***()**

**else:**

result **=** self**.***collection***.***find***(**filter\_query**)**

**return** **list(**result**)**

**except** **Exception** **as** e**:**

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

**return** **[]**

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

"""Updates document(s) from a specified MongoDB database and specified collection."""

**try:**

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

**return** result**.***modified\_count*

**except** **Exception** **as** e**:**

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

**return** 0

**def** delete**(**self**,** query**):**

"""Removes document(s) from a specified MongoDB database and specified collection."""

**try:**

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

**return** result**.***deleted\_count*

**except** **Exception** **as** e**:**

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

**return** 0

### Tests

**IPYNB with filter queries**

# Setup the Jupyter version of Dash

**from** jupyter\_dash **import** JupyterDash

# Configure the necessary Python module imports for dashboard components

**import** dash

**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

#### FIX ME #####

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

**from** animal\_shelter **import** AnimalShelter

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

# Data Manipulation / Model

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

# FIX ME update with your username and password and CRUD Python module name

username **=** "aacuser"

password **=** "hockey4"

host **=** 'nv-desktop-services.apporto.com'

port **=** 32418

database **=** "AAC"

collection **=** "animals"

# 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***({}))**

# MongoDB v5+ is going to return the '\_id' column and that is going to have an

# invlaid object type of 'ObjectID' - which will cause the data\_table to crash - so we remove

# it in the dataframe here. The df.drop command allows us to drop the column. If we do not set

# inplace=True - it will reeturn a new dataframe that does not contain the dropped column(s)

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

## Debug

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

# print(df.columns)

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

# Dashboard Layout / View

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

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

#FIX ME Add in Grazioso Salvare’s logo

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

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

#FIX ME Place the HTML image tag in the line below into the app.layout code according to your design

#FIX ME Also remember to include a unique identifier such as your name or date

app**.***layout* **=** html**.***Div***([**

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

html**.***Center***(**html**.***B***(**html**.***H1***(**'Animal Shelter Dashboard'**))),**

html**.***Center***(**html**.***B***(**html**.***H2***(**'Chad Nadeau'**))),**

html**.***A***(**

html**.***Img***(**

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

style**={**'height'**:** 350**,**

'width'**:** 350**,**

'display'**:** 'block'**,**

'margin-left'**:** 'auto'**,**

'margin-right'**:** 'auto'

**},**

alt**=**"Loading..."

**),**

href**=**'https://snhu.edu'

**),**

html**.***Hr***(),**

dcc**.***RadioItems***(**

**id=**'filter-type'**,**

options**=[**

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

**{**'label'**:** 'Mountain'**,** 'value'**:** 'Mountain'**},**

**{**'label'**:** 'Disaster'**,** 'value'**:** 'Disaster'**},**

**{**'label'**:** 'Reset'**,** 'value'**:** 'Reset'**}**

**],**

value**=None,**

labelStyle**={**'display'**:** 'inline-block'**}**

**),**

html**.***Br***(),**

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**.***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'**,**

**)**

**]**

**),**

html**.***Hr***(),**

**])**

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

# Interaction Between Components / Controller

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

*@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

**if** filter\_type **is** **None:**

**return** df**.***to\_dict***(**'records'**),** **[{**"name"**:** i**,** "id"**:** i**,** "deletable"**:** **False,** "selectable"**:** **True}** **for** i **in** df**.***columns***]**

**if** filter\_type **==** 'Reset'**:**

filter\_query **=** **{}**

**elif** filter\_type **==** 'Water'**:**

filter\_query **=** **{**

'breed'**:** **{**'$in'**:** **[**

'Labrador Retriever Mix'**,**

'Chesa Bay Retr Mix'**,**

'Newfoundland Mix'**,**

'Newfoundland/Labrador Retriever'**,**

'Newfoundland/Australian Cattle Dog'**,**

'Newfoundland/Great Pyrenees'

**]},**

'sex\_upon\_outcome'**:** 'Intact Female'**,**

'age\_upon\_outcome\_in\_weeks'**:** **{**'$gte'**:** 25**,** '$lte'**:** 157**}**

**}**

**elif** filter\_type **==** 'Mountain'**:**

filter\_query **=** **{**

'breed'**:** **{**'$in'**:** **[**

'German Shepherd'**,**

'Alaskan Malamute'**,**

'Old English Sheepdog'**,**

'Siberian Husky'**,**

'Rottweiler'

**]},**

'sex\_upon\_outcome'**:** 'Intact Male'**,**

'age\_upon\_outcome\_in\_weeks'**:** **{**'$gte'**:** 25**,** '$lte'**:** 157**}**

**}**

**elif** filter\_type **==** 'Disaster'**:**

filter\_query **=** **{**

'breed'**:** **{**'$in'**:** **[**

'Doberman Pinsch'**,**

'Doberman Pinsch Mix'**,**

'German Shepherd'**,**

'Golden Retriever'**,**

'Golden Retreiver Mix'**,**

'Bloodhound'**,**

'Rottweiler'

**]},**

'sex\_upon\_outcome'**:** 'Intact Male'**,**

'age\_upon\_outcome\_in\_weeks'**:** **{**'$gte'**:** 19**,** '$lte'**:** 301**}**

**}**

filtered\_data **=** db**.***read***(**filter\_query**)**

filtered\_df **=** pd**.***DataFrame***.***from\_records***(**filtered\_data**)**

filtered\_df**.***drop***(**columns**=[**'\_id'**],** inplace**=True)**

columns **=** **[{**"name"**:** i**,** "id"**:** i**,** "deletable"**:** **False,** "selectable"**:** **True}** **for** i **in** filtered\_df**.***columns***]**

data **=** filtered\_df**.***to\_dict***(**'records'**)**

**return** data**,** columns

*@app***.***callback***(**

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

**[**Input**(**'datatable-id'**,** "data"**),**

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

**def** update\_graphs**(**data**,** filter\_type**):**

**if** filter\_type **is** **None** **or** filter\_type **==** 'Reset'**:**

**return** **[]**

dff **=** pd**.***DataFrame***.***from\_records***(**data**)**

**if** **not** dff**.***empty***:**

names **=** dff**[**'breed'**].***value\_counts***().***keys***().***tolist***()**

values **=** dff**[**'breed'**].***value\_counts***().***tolist***()**

**return** **[**

dcc**.***Graph***(**

figure **=** px**.***pie***(**

names **=** names**,**

values **=** values**,**

color\_discrete\_sequence**=**px**.***colors***.***sequential***.***RdBu***,**

width**=**800**,**

height**=**500

**)**

**)**

**]**

**else:**

**return** **[]**

#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** index **is** **None:**

row **=** 0

**else:**

row **=** index**[**0**]**

# Austin TX is at [30.75,-97.48]

**return** **[**

dl**.***Map***(**style**={**'width'**:** '800px'**,** 'height'**:** '500px'**},** center**=[**30.75**,-**97.48**],** zoom**=**10**,** children**=[**

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

# Marker with tool tip and popup

# Column 13 and 14 define the grid-coordinates for the map

# Column 4 defines the breed for the animal

# Column 9 defines the name of the animal

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**])**

**])**

**])**

**])**

**]**

**print(**'Connection Successful!'**)**

app**.***run\_server***(**debug**=True)**

### Video of Functionality.

**

## Contact

Chad Nadeau

SNHU CS-340