# CS 340 README Template

## About the Project/Project Title

This project incorporates the use of CRUD (create, read, update and delete) into a Python module that allows the user to connect to a database from the client side. It also serves as a means to test that connection by writing a test script as input from the user to the database. Additionally, I have added the Dash frame work completed in Jupyter notebook to run the application using the CRUD module that was already created.

## Motivation

CRUD functionality is a necessary component to any database. A user can access the data stored within the database, create new data to be stored within the database. The user can query the database to read through the existing data that is already stored to find something specific, and return that find to the user. The user can also update data that is stored in the database as well as delete it from the database. All these interactions are incredibly important and necessary to have a fully working database. The way that this is incorporated and more importantly tested, is by using a Python module. This ensures that we have the correct connection and that the database is successfully taking user input and displaying back to the user the appropriate responses. We also use the Python module to incorporate the Dash framework with the use of the completed and fully tested CRUD module.

## Getting Started

To get a local copy up and running, follow these simple example steps:

Step 1: Once you have started mongo and are back in the Linux command prompt and out of the Mongoshell, verify that you are in the correct file system within the Linux command prompt, import the file you want to use into the database. Now, check that it has successfully imported into the correct database. Below is a screen shot of that step with the code necessary to import and confirm import was successful. \*\* Please note, the port number will be your port number that is given to you when you start mongo. \*\*

You will also need to set up user authentication in this step as it will be necessary to test user access to the database through authentication. This can be done using the db.createUser() function inside the Mongoshell once you are in the correct file system you would like to be inside the database. You can confirm you have successfully created a user by using the getUsers() function, also inside the Mongoshell.

Step 2: Open Jupyter Notebook and create a PY file to create the Create and Read functionality for the database. In this case, shown below, I used the name “animal\_shelter.py”. This is where you will create your CRUD functionality for the class structure and define the authentication as well as how to create/insert data into the database as well as read the data in the database, update the data as well as delete the data.

Step 3: Create a IPYNB file within Jupyter Notebook. This is where we will write a script to test the functionality of our PY file that we created in Step 2. In this case, we are testing the creation of the object from the class that we created, to ensure that we successfully gain access to the database by providing user input for username and password to test that our authentication is functioning correctly. In addition to testing the authentication, we will also need to test the create, read, update and delete data functions as well. We can confirm this is functioning with simple print displays. Once the code is run from the IPYNB file, we can see the print statements that are displayed successfully. We can also confirm in the mongo shell that we are adding to the database, as well as updating and deleting.

Step: 4 Create another IPYNB file within Jupyter Notebook. This is where we will create the Dash frame work with view and controller structures to incorporate our previous CRUD module and allow a user to have a fully functioning web application that allows for user input, in my case I used radio buttons to sort through the database of existing animals to find specific breeds within certain age frames and of a required sex for individual rescue missions.

## Installation

The tools that are needed here, are a Linux command prompt, a database with available space for files to be stored, as well as Jupyter Notebook to run the PY and IPYNB files.

Here is the link with steps to install Jupyter Notebook: <https://jupyter.org/install>

MongoDB is used as the model component for the development of these modules while interfacing with Python because MongoDB and Python have a lot of similar characteristics as well as similar syntax so it is easier to understand, and work with.

## Usage

### Code Example

animal\_shelter.py:

from pymongo import MongoClient

from bson.objectid import ObjectId

import json

class AnimalShelter(object):

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

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

# Initializing the MongoClient. This helps to

# access the MongoDB databases and collections.

#self.client = MongoClient('mongodb://localhost:33549')

## with authentication

self.client = MongoClient('mongodb://%s:%s@localhost:33549/?authMechanism=DEFAULT&authSource=AAC' % (username, password))

# where xxxx is your unique port numberS

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

# Complete this create method to implement the C in CRUD.

def create(self, data):

if data is not None:

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

return True

else:

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

# Created a method to read the files and find all of the data, and not just one as I did below.

def read(self,data):

cursor = self.database.animals.find(data, {'\_id':False})

return cursor

# Create method to implement the R in CRUD.

def read\_all(self, data):

if data is not None:

global globalRead

globalRead = self.database.animals.find\_one(data, {'\_id':False})

return globalRead

else:

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

# Complete this create method to implement the U in CRUD.

def update(self, data, update\_info, data\_test):

if data is not None:

localUpdate = self.database.animals.update\_one(data, update\_info)

global globalUpdate

globalUpdate = self.database.animals.find\_one(data\_test, {'\_id':False})

return globalUpdate

else:

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

# Complete this create method to implement the D in CRUD.

def delete\_one(self, data\_Delete):

if data\_Delete is not None:

global globalDelete

globalDelete = self.database.animals.delete\_one(data\_Delete)

return globalDelete

else:

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

# Returning global read value

def globalReadFunc(self):

return globalRead

# Returning global update value

def globalUpdateFunc(self):

return globalUpdate

# Returning global delete value

def globalDeleteFunc(self):

return globalDelete

***ProjectTwoDashboard.ipynb***

*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 base64*

*#### FIX ME #####*

*# changed animal\_shelter and AnimalShelter to match my CRUD Python module file name and class name*

*from animal\_shelter import AnimalShelter*

*###########################*

*# Data Manipulation / Model*

*###########################*

*#username and password and CRUD Python module name*

*username = "aacuser"*

*password = "aacuser"*

*shelter = AnimalShelter(username, password)*

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

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

*#########################*

*# Dashboard Layout / View*

*#########################*

*app = JupyterDash('SimpleExample')*

*#Added 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())*

*app.layout = html.Div([*

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

*#Placed the HTML image tag in the line below into the app.layout code*

*#Added in the image and centered it on the page*

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

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

*# Added a unique identifier with my name*

*html.Center(html.H2('Breunna Bingham')),*

*html.Center(html.H3('Animal Dashboard')),*

*html.Hr(),*

*html.Div(*

*# Added in code for the interactive filtering options using Radio Buttons*

*dcc.RadioItems(*

*id = 'rescue\_type',*

*options=[{'label': 'Water Rescue', 'value':'water'},*

*{'label': 'Mountain/Wilderness Rescue', 'value':'mountain'},*

*{'label': 'Disaster/Individual Rescue', 'value':'disaster'},*

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

*],*

*value = 'reset',*

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

*)*

*),*

*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'),*

*# Set up the features for interactive data table to make it user-friendly for the client*

*# copied my code from module six*

*editable=False,*

*sort\_action="native",*

*sort\_mode="multi",*

*filter\_action="native",*

*column\_selectable=False,*

*row\_selectable="single",*

*row\_deletable=False,*

*selected\_columns=[],*

*selected\_rows=[],*

*page\_action="native",*

*page\_current= 0,*

*page\_size= 10,*

*),*

*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(*

*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('rescue\_type','value')])*

*def update\_dashboard(value):*

*# Added code to filter the interactive data table with MongoDB queries*

*# according to the specifications from the client*

*# shows all animals unfiltered*

*if (value == 'reset'):*

*df = pd.Dataframe.from\_records(shelter.read({}))*

*# Shows only the intact female breeds listed under water rescue within a certain age frame*

*elif (value == 'water'):*

*df = pd.DataFrame.from\_records(shelter.read({"breed": {"$in": ["Labrador Retriever Mix",*

*"Chesapeake Bay Retriever",*

*"Newfoundland"]},*

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

*"age\_upon\_outcome\_in\_weeks":{"$gte":26}, #Greater than/ equal to 26 weeks*

*"$and": [{"age\_upon\_outcome\_in\_weeks":{"$lte":156}}]})) #less than/ equal to 156 weeks*

*# Shows only the intact male breeds listed under mountain rescue within a certain age frame*

*elif (value == 'mountain'):*

*df = pd.DataFrame.from\_records(shelter.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}, #Greater than/ equal to 26 weeks*

*"$and": [{"age\_upon\_outcome\_in\_weeks":{"$lte":156}}]})) #less than/ equal to 156 weeks*

*# Shows only the intact male breeds listed under disaster rescue within a certain age frame*

*elif (value =='disaster'):*

*df = pd.DataFrame.from\_records(shelter.read({"breed":{"$in":["Doberman Pinscher",*

*"German Shepherd",*

*"Golden Retriever",*

*"Bloodhound",*

*"Rottweiler"]},*

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

*"age\_upon\_outcome\_in\_weeks":{"$gte":20}, #Greater than/ equal to 20 weeks*

*"$and": [{"age\_upon\_outcome\_in\_weeks":{"$lte":300}}]})) #less than/ equal to 300 weeks*

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

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

*return (data,columns)*

*@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]*

*@app.callback(*

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

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

*def update\_graphs(viewData):*

*# Added the code needed to display a pie chart of available breeds*

*df = pd.DataFrame.from\_dict(viewData)*

*return [*

*dcc.Graph(*

*figure = px.pie(df, names = 'breed')*

*)*

*]*

*@app.callback(*

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

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

*Input('datatable-id', 'derived\_viewport\_selected\_rows')])*

*def update\_map(viewData, selected\_rows):*

*#FIXME Add in the code for your geolocation chart*

*viewDF = pd.DataFrame.from\_dict(viewData)*

*dff = viewDF.loc[selected\_rows]*

*return [*

*dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.75,-97.48], zoom=15, children=[*

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

*# Marker with tool tip and popup*

*dl.Marker(position=[dff.loc[0,'location\_lat'],dff.loc[0,'location\_long']], children=[*

*dl.Tooltip(dff.loc[0,'breed']),*

*dl.Popup([*

*html.H1("Animal Name"),*

*html.P(dff.loc[0,'name'])*

*])*

*])*

*])*

*]*

*app*

### Tests

When writing the test script you want to make sure you are fully testing the entire capability of the PY file. In this case, we had to test for authentication, to gain access into the database using a preexisting account that was created previously in Mongo. We also tested the create functionality of CRUD by allowing the user to input new data that they wanted stored in the database. This is done by using a simple if/else statement, where the system Prints out a message if it was successful or not. The same is done to test the functionality of the Read statement used to allow the user to query data that is already stored in the database. If found, then the system issues a print statement to the user notifying them that the system was successful. For the update function we will update one of the data files stored within the collection of the database, and then we will use the find\_one function to confirm the information we updated was successfully reflected when querying from the database. For the delete functionality, we will use the delete\_one function and pass in parameters to make sure we are deleting a specific file from the database. In this case it was the dog file that we initially added, updated and now are deleting.

To test theProjectTwoDashboard.ipynb file I had to incorporate the queries that would allow the user to select a radio button and choose which rescue type they are looking for. Once they selected a rescue type, the CRUD module would use the read function and find all of the dogs that matched the specified breed, sex and age criteria per rescue type. One that was in place, I was able to add in the company logo on the application, as well as populate a map with a cursor pointing to the location of the animal using latitude and longitude coordinates stored within the database. I also populated a pie chart representing each of the breeds available and the percentage they make up for that specific rescue type. There were many challenges that were encountered during this project. The main problem that I encountered the most was while I was using Jupyter Notebook. My ipynb file every time I ran it would freeze consistently but at inconsistent times. For example, I could click one radio button and I would have the whole system freeze and not respond to any other user input i.e., selecting other radio buttons. This caused me to have to close out the file and reopen it every time I would test my code for functionality. This also put a damper on determining if my code was problematic or if it was the application, Jupyter Notebook that was problematic. I also encountered a problem with the Water rescue query. Each time I ran the code, I noticed that when I selected the radio button representing the Water Rescue type, that no dogs appeared yet I had three different breeds within the query. I finally figured out that my query had listed Labrador with a lower-case L instead of the uppercase L as it is listed in the Database. Once that changed, I ran the code again and realized that the reason the other two breeds didn’t show up was because there was no criteria that matched those breeds so none of them populated, only Labradors did.

### Screenshots

Inserting a file into database to use for CRUD functionality, and creating a user and verifying user has been created successfully:

*Text

Description automatically generated*

Text

Description automatically generated

A screenshot of a computer

Description automatically generated

*animal\_shelter.py:*

*A picture containing table

Description automatically generated*

*test\_script.ipynb:*

*Text

Description automatically generated with low confidence*

*ProjectTwoDashboard.ipynb*

*Snip 1*

*A picture containing text

Description automatically generated*

*Snip 2*

*Text

Description automatically generated with medium confidence*

*Snip 3*

*A picture containing timeline

Description automatically generated*

*Snip 4*

*Graphical user interface, text, application

Description automatically generated*

*Step by step process of running each of the CRUD functions within the test script and verifying each step within the mongoshell as well.*

*CREATE:*

*Performed a search within Mongo shell to confirm Dog had been added and can be found within the database.*

*Graphical user interface, application

Description automatically generated*

*READ:*

*Queried to make sure that the dog added reflected within the database to confirm my read function was working correctly.Graphical user interface, application

Description automatically generated*

UPDATE:

You can see that the years updated for the Dog that I just added and queried.Graphical user interface, application

Description automatically generated

DELETE:

Performed a find\_one search within Mongoshell to confirm the dog could not be found and therefore was deleted. Also wanted to note here, It didn’t make sense to use a JSON format output since the dog had been removed and there would be nothing to query and print out and would result in an Error message that the dog could not be deleted from the test script.

Graphical user interface, text, application

Description automatically generated

CRUD – Running all of the functions together Create, Read, Update and Delete:

As you can see the query resulted in null multiple times after running the script since the dog file was deleted after it had been created, read, and updated.

Graphical user interface, application

Description automatically generated

Now testing the functionality of the CRUD module with the Dash Framework:

This is the code running on the separate window.

Note\*\* I could not get the map and pie chart to populate in the separate window but they would populate in the window from the .ipynb file. It was harder to see the logo and the results in the small window so I wanted to show these queries in these screen shots and a separate smaller window proving the map and pie chart was working as well, without seeing the logo in view, etc.

Running the code initially querying all animal data from the database

A picture containing text

Description automatically generated

## Selecting Water Rescue radio button to query only the selected breed, sex and age criteria that met the required conditions.

A picture containing table

Description automatically generated

Selecting Mountain Rescue radio button to query only the selected breed, sex and age criteria that met the required conditions

A picture containing text

Description automatically generated

Selecting Disaster Rescue radio button to query only the selected breed, sex and age criteria that met the required conditions

A picture containing text

Description automatically generated

Showing a small snip of the map and the pie chart that populates ( again, I couldn’t include the logo and even the table in these snips if I wanted to show the pie chart and map. They would never populate on the separate window option.)

Water Rescue:

Graphical user interface, application

Description automatically generated

Mountain Rescue:

Graphical user interface

Description automatically generated with medium confidence

Disaster Rescue

Graphical user interface

Description automatically generated with medium confidence

## Contact

Breunna Bingham