Anthony Vigil

**Project Two: README**

CS-340: Client/Server Development

Professor: Jeff Sanford

# CS 340 README – Anthony Vigil

## About the Project/Project Title

The title of the project is Create and Read in Python. The objective of this project is to create and read functionality in Python to help execute a Mongo job. It will return combinations of the sex, breed and age to train the rescue dogs.

## Motivation

This project is meant to make things easier when querying the Mongo Database. The csv document in uploaded into the database using the Python-friendly interface. It is implemented by a created Python script which runs to connect the Mongo Server and initializes the CRUD class for the end user.

**Motivation for Using Dash**

Dash is used as the tool to build the dashboard. It is used due to its versatility and outcomes for features. Dash is based off JavaScript and uses HTML tags for outputs. The updates on inputs in the application callbacks are executed with their respective instructions.

## Getting Started

Making the program requires several steps. First, it begins with creating a Mongo database; Second, it requires the program CRUD Python file to be utilized to access and read the database. The last step involves creating the Dash web application to read all the information from the database in clear tables or charts.

1. Create the Mongo Database and create the database as AAC.
2. Create a user with privileges of read/write to enable access to the AAC database

The Database will then be authenticated by using a created username and password for the Admin. I created a user with the Read and Write privilege to be used for accessing the database. I logged into Mongo using the created username and password.

![Text

Description automatically generated]()

Import the data or information from the aac\_shelter\_outcomes.csv file provided

![Text

Description automatically generated]()

![Text

Description automatically generated]()

2. The port number will need to be updated in the Animal\_Shelter python code.

a. It is very important for the port number to be updated.

3. The “aacuser” and “Password” must be updated in the username and password sections of the code.

a. Snapshot of the operation

![Text

Description automatically generated]()

4. Open the Jupyter notebook with test code and test the functions by adding and deleting.

5. Create the Dash web application by configuring the HTML/CSS layout and testing for the correct data.

6. Create the app callback for all the data.

7. Program the database queries with the options given from the client.

a. Combine searches together to test the accuracy.

8. Create the application callback for updating the map when the user clicks on a category while also creating a program to test the functionality.

a. For this part, I created the test a bit more complicated due to an error in reading the data from the database.

b. Filter out the rows but carefully due to the variables used in the Dash such as “supplied\_virtual\_row\_ids”. Integrating the “selected\_rows” is easier to gather information.

9. Create a chart, I selected a pie chart, based on the data on the screen for the dashboard. Using the pie chart, create a callback that will update the chart with the data when it is filtered.

a. Creating this produced several problems due to its combination of the data. The array that would read the information from the file would not format correctly so changing the code a bit helped arrange it better. By having an index of names and array of numbers, it will allow the data to display accurately.

## Installation

The tools used were the Jupyter Notebook, command line and the libraries in pymongo and Objectid.

**Jupyter Notebooks:**

Jupyter can be installed using the command line in your operating system; following the instructions in this page: <https://jupyter.org/install>. For the assignment, I used Apporto to utilize Jupyter.

**Python:**

Installing Python can be done by following the instruction in the given website: <https://realpython.com/installing-python/>. When the installation is complete, the terminal can be utilized from your windows system

**MongoDB:**

MongoDB comes in community edition. Instructions on downloading or installing can be found here: <https://www.mongodb.com/docs/manual/installation/>

**Plotly:**

Plotly is imported into the Dash to generate the chart or tables used. It is known as a charting tool.

<https://plotly.com/python-api-reference/generated/plotly.express.pie.html>

**Dash:**

Dash is known as the framework that is used to create the web applications. It is imported using the Dash Core Components in Jupyter Notebook. More information on Dash is available here: <https://pypi.org/project/dash/>

**Pandas:**

Pandas is also known for creating web applications; It is a tool that is combined with Python to create data frames. <https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html>

## Usage

The project can be used to access the data that was created and also read from MongoDB, that was imported. The application is used for 3 functions such as radial buttons for user sorting, resetting information, and checking for desired filters. The user is able to click on the radial buttons to search different types of rescues in the database. The reset will change the table back to the original state where all the animals are unfiltered. The map will generate with the updates once the buttons or filters are clicked from the user. A pie chart will also be generated to show the statistics of the breeds. The application utilizes four functions: create, read, update, and delete. The first two functions will create the data and read it into the table. The update will search for any current animal to change its status. The delete function will delete any animal that is prompted by the user.

### Code Example

**Reset All Animals**

A picture containing application

Description automatically generated

**Mountain Rescue:**

Graphical user interface, application, table

Description automatically generated

**Disaster Rescue:**

Graphical user interface, application

Description automatically generated with medium confidence

**Water Rescue:**

Graphical user interface, application

Description automatically generated

**Code Images**

***Read Data and App Layout:***

**![Graphical user interface, text, application

Description automatically generated]()**

***App Callback:***

**![Text

Description automatically generated]()**

***Create Map:***

**![Text

Description automatically generated with medium confidence]()**

**Python Code:**

**![A picture containing text

Description automatically generated]()**

**Code: (CRUD)**

**Create:**

#C operation for C in CRUD

def create(self, data):

#use try/except block for boolean processing

try:

if data is not None:

#print(type(data)) Confirm data

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

pprint(insert\_result)

#print("True")

return True #return value

else:

# error message

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

except:

#print("False")

return False #return value

**Read:**

#R operation for R in CRUD

def read(self, target):

# try/except block for testing in the unit tests

try:

if target is not None:

#print(type(target)) # dictionary

read\_result = list(self.database.animals.find(target, {"\_id": False}))

#pprint(read\_result) # displays results

return read\_result

else:

#lets the user know there was a problem

raise Exception("Nothing to search, because the target parameter is empty")

return False

except Exception as e:

print("An exception occurred: ", e)

**Update:**

#U operation for U in CRUD

def update(self, fromTarget, toTarget, count):

if fromTarget is not None:

if count == 1:

update\_result = self.database.animals.update\_one(fromTarget, toTarget)

pprint("Matched Count: " + str(update\_result.matched\_count) + ", Modified Count: " + str(update\_result.modified\_count))

if update\_result.modified\_count == 1:

print("Success!")

print(update\_result)

return True

else:

print("Something went wrong")

return False

elif count == 2:

update\_result = self.database.animals.update\_many(fromTarget, toTarget)

pprint("Matched Count: " + str(update\_result.matched\_count) + ", Modified Count: " + str(update\_result.modified\_count))

if update\_result.modified\_count == update\_result.matched\_count:

print("Success!")

print(update\_result)

return True

else:

print("Something went wrong, all items matching the target may not have been updated. Run a search to verify")

print(update\_result)

return True

else:

print("Count not recognized - try again.")

return False

else:

#lets the user know there was a problem

raise Exception("Nothing to update, because at least one of the target parameters is empty")

return False

**Delete:**

def deleteData(self, target, count):

if target is not None:

if count == 1:

try:

delete\_result = self.database.animals.delete\_one(target)

pprint("Deleted Count: " + str(delete\_result.deleted\_count))

if delete\_result.deleted\_count == 0:

print("Nothing to be deleted using the target data.")

print(delete\_result)

return True

else:

print("Success!")

print(delete\_result)

return True

except Exception as e:

print("An exception has occurred: ", e)

elif count == 2:

try:

delete\_result = self.database.animals.delete\_many(target)

pprint("Deleted Count: " + str(delete\_result.deleted\_count))

if delete\_result.deleted\_count == 0:

print("Nothing to be deleted using the target data.")

print(delete\_result)

return True

else:

print("Success!")

print(delete\_result)

return True

except Exception as e:

print("An exception has occurred: ", e)

return False

else:

print("Count not recognized - try again.")

return False

else:

#lets the user know there was a problem

raise Exception("Nothing to delete, because the target parameter is empty")

return False

**Roadmap & Features**

The project allows for 4 to 5 items to be selected but with some updating it can allow more. Updating the map will show more animals by resetting the data. The pie chart would be more efficient if there was a menu apart from the top radio items to filter the animals.

**Known Issues:**

The main issue during the project was reading the information from the data sheet. The issue is the authentication would fail and another user would need to be created to access the information for further use. I created about 7 users throughout the process to test the functionality but ultimately the filtering of the dictionary into indexes and the array was the most difficult. The terminal was used most of the time to create the users and check for authentication.

**Known Issues (Dash):**

I had difficulty checking the functionality of the rows when clicking on them in the left side box. It would output the row but adding more than 2 would not populate the map. It required some code editing to fix the issue but it can be more efficient by debugging and testing more.

## Contact

Your name: Anthony Vigil

anthonyvigil@snhu.edu