# CS 340 Anatomy of a README

## About the Project/Project Title

The purpose of the project is to develop a program for Grazioso Salvare that creates an interactive web portal using Dash framework in Python to filter through the Austin Animal Center Mongo Database.

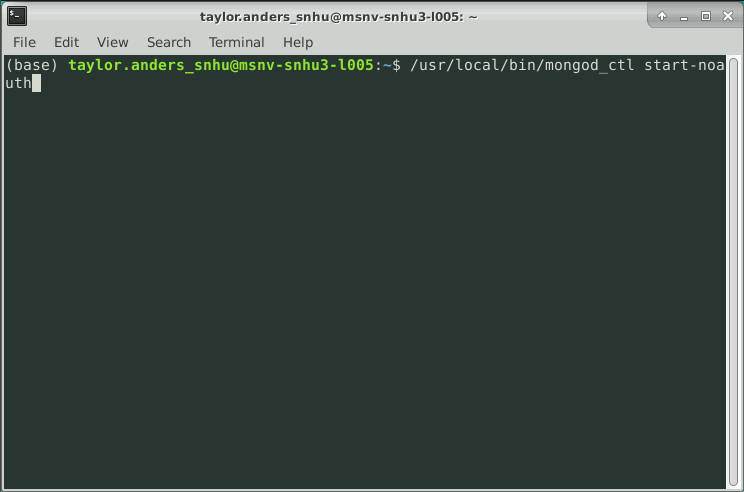
## Motivation

The purpose of this project is to create a program that makes it easier for working with the database in Mongo. Entering in command prompts is time consuming and moderately difficult because command prompts are entered line by line. There is little room for corrections of typos, and it is easy to get lost between steps. Building a web-based platform gives the user the ability to search through this database with relative ease, increasing usefulness and productivity.

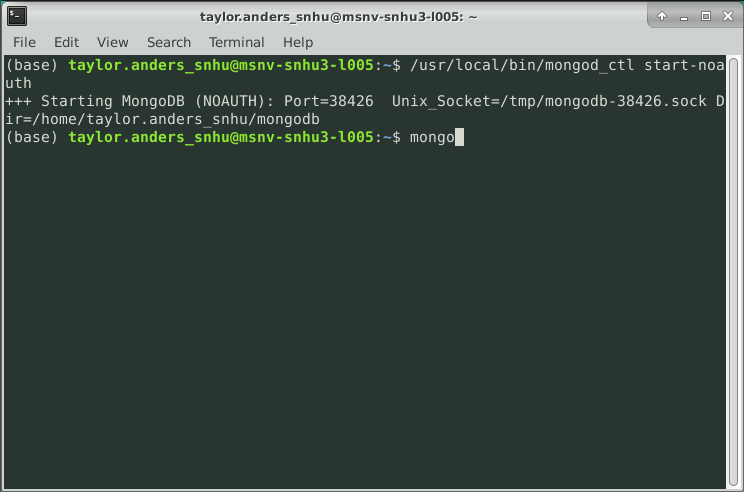
## Getting Started

The first thing we need to do is make sure we have Mongo installed. Please refer to the installation section and come back here after that is done. Once that is done, let’s open up the command prompt and establish a link to MongoDB. This can be done using the syntax:

/usr/local/bin/mongod\_ctl start-noauth



After that enter ‘mongo’ in the command prompt to start.



Once you are logged in, we can switch to Jupyter Notebook. If you have a user authentication step, this will be covered later.

## Installation

We need several programs for this to run properly. First, we need to install Python. Python can be found at <https://www.python.org/downloads/> Search for your system and select the proper version. Next, we need Jupyter Notebook. That program can be found here: <https://jupyter.org/install> Once that is installed you will need to install PyMongo. Follow the documentation here: <https://pymongo.readthedocs.io/en/stable/installation.html>. Finally, when those are all installed, we can get to work!

## Usage

Dash Python in conjunction with PyMongo is a powerful tool that can be used to do a multitude of amazing things. In this application we are going to build a web portal that can access a Mongo Database through Python and return a simple interface using Dash that we can interact with the database to sort and filter our results.

As mentioned previously, if you have to authenticate a user for your database we will follow a few separate steps. First we need to again open the command prompt. Instead of using:

/usr/local/bin/mongod\_ctl start-noauth

We will use:

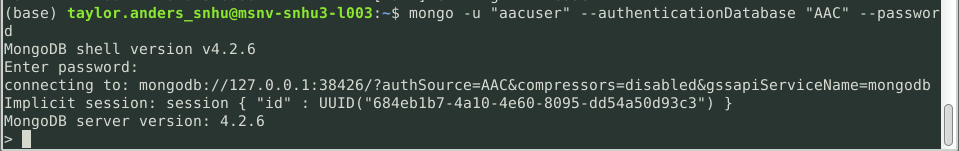
/usr/local/bin/mongod\_ctl start



Then we will login using the syntax:

Mongo – u “userlogin” –authenticationDatabase “database” –password

Keep in mind, the “userlogin” will be your login and “database” is the database to which your user is assigned. Our example user is “aacuser” and our database is the AAC database. It would look like this.



You will then be prompted to enter your password. Keep in mind that the password will not be displayed. This is normal. Once in, we can work in Jupiter.

### Code Example

Starting on our project, we will upload out CRUD method Python file. CRUD stands for Create, Read, Update, Delete. This is an important tool in working with databases. The syntax for each individual CRUD can vary but ours looks like this:

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://%s:%s@localhost:38426/AAC' % (username, password))

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

# Method to implement the C in CRUD.

def create(self, data):

if data is not None:

# Insert the data, and return true if successful.

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

return True

else:

# Data was invalid.

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

return False

# Method to implement the R in CRUD.

def read(self, data):

if data is not None:

# Return all records that match.

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

else:

# Data was invalid.

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

return False

# Method to implement the U in CRUD.

def update(self, search, data):

if data is not None:

# Update the chosen data in the animal database.

updateRecord = self.database.animals.update\_many(search, data)

oldFor = json.dumps(search) # json.dumps converts library to string for outputting readability

newFor = json.dumps(data)

pStr = 'The data ' + oldFor + ' was successfully changed to ' + newFor

return pStr

else:

# Data is invalid.

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

return False

# Method to implement the D in CRUD.

def delete(self, data):

if data is not None:

# Delete the specified data in the animal database.

self.database.animals.delete\_many(data)

delDataFor = json.dumps(data)

pStr = 'The data ' + delDataFor + ' was successfully deleted.'

return pStr

else:

# Data is invalid

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

return False

A screenshot of this file in Jupyter notebook will look like this:



Next, we will start working with Dash. Dash is a powerful framework that allows us to access all sorts of customizable interfaces. From interaction and visualization to full stack web apps. The documentation for Dash is quite extensive and their entire library can be found here: <https://plotly.com/>

While working with Dash for our application, we will be focusing on just a few of the many applications. First will be the Dash HTML Components, found here: <https://dash.plotly.com/dash-html-components> Dash Datatables, found here: <https://dash.plotly.com/datatable> Callbacks, found here: <https://dash.plotly.com/basic-callbacks> and <https://dash.plotly.com/advanced-callbacks> Radio buttons found here: <https://dash.plotly.com/dash-core-components/radioitems> And finally their pie charts, found here: <https://plotly.com/python/pie-charts/>

Again, these are just a few of the many uses and there is quite a bit more you will be able to do with Dash, but these are the necessary components for out project. Out code build in Jupyter notebook will looks like this:

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 #####

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

from \_\_main\_\_ import AnimalShelter

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

# Data Manipulation / Model

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

# FIX ME change for your username and password and CRUD Python module name

username = "aacuser"

password = "9876"

shelter = AnimalShelter(username, password)

# Class read method must support return of cursor object

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

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

# Dashboard Layout / View

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

app = JupyterDash('Project Two')

# 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'}),

# Centered photo

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

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

html.Hr(),

# FIXME Add in code for the interactive filtering options. For example, Radio buttons, drop down, checkboxes, etc.

html.Div(

# Filter Radio buttons from https://dash.plotly.com/dash-core-components/radioitems

dcc.RadioItems(

id='filter-type', options=[

{'label':'Water Rescue','value':'btn1'},

{'label':'Mountain or Wilderness Rescue','value':'btn2'},

{'label':'Disaster Rescue or Individual Tracking','value':'btn3'},

{'label':'Reset','value':'btn4'}

],

value='btn4',

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'),

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

# If you completed the Module Six Assignment, you can copy in the code you created here

# sorting features to use from https://dash.plotly.com/datatable/interactivity

editable=False,

filter\_action='native',

sort\_action='native',

sort\_mode='multi',

column\_selectable=False,

row\_selectable='single',

row\_deletable=False,

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',

)

]),

html.Br(),

'Taylor Anderson SNHU CS-340 Project Two'

])

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

# 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 absto filter interactive data table with MongoDB queries

if filter\_type == 'btn1':

df = pd.DataFrame.from\_records(shelter.read({"animal\_type":"Dog",

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

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

"age\_upon\_outcome\_in\_weeks":{"$gte":20},

"age\_upon\_outcome\_in\_weeks":{"$lte":300}}))

elif filter\_type == 'btn2':

df = pd.DataFrame.from\_records(shelter.read({"animal\_type":"Dog",

"breed":{"$in":["German 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 == 'btn3':

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

elif filter\_type == 'btn4':

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

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):

###FIX ME ####

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

# Pie chart from https://plotly.com/python/pie-charts/

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

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

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

return [

dcc.Graph(

figure = px.pie(data\_frame=dff,values=values,names=names,height=500,width=500)

)

]

@app.callback(

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

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

def update\_map(viewData):

# FIXME: Add in the code for your geolocation chart

# If you completed the Module Six Assignment, you can copy in the code you created here.

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

# 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"),

# Marker with tool tip and popup

dl.Marker(position=[30.75,-97.48], children=[

dl.Tooltip(dff.iloc[0,4]),

dl.Popup([

html.H1("Animal Name"),

html.P(dff.iloc[1,9])

])

])

])

]

app

A screenshot of the code is below:

Text, letter

Description automatically generatedA picture containing text

Description automatically generated

### Output

The above code will output a cell that can be executed in a new window. A screenshot of this window is below showing the output for selecting the radio buttons.

The first is the default and takes in the entire database. That will looks like this:

A picture containing calendar

Description automatically generated

Following that, the first radio button selected which filters for dogs suited for Water Rescue will return the screen below:

Calendar

Description automatically generated

After that, the next radio button filters for Dogs suited for Mountain or Wilderness Rescue and will look like this:

A picture containing graphical user interface

Description automatically generated

And finally, the last radio button folders for Dogs suited for Disaster or Individual Tracking:

A picture containing text

Description automatically generated

As you can see, using these components can give you an amazing looking and powerful tool. And all of this is just the tip of the iceberg. MongoDB, Dash, Python, and Jupyter notebook can be used in numerous ways that can be tailored to nearly any scenario. Now go create something amazing!

## Roadmap/Features (Optional)

Known Issues:

It is important to note that when working with Mongo databases you need to have MongoDB open. This in and of itself can cause some issue as working with command prompts may be a bit above the average users head. The second known issue is that it is highly dependent on the internet connection. While this is not always an issue, slower internet traffic can cause a delay in some of the updating. Any further questions can be directed to:

## Contact

Taylor Anderson

Taylor.anderson5@snhu.edu