CS 340: Project Two README

Spencer Hayden

Southern New Hampshire University

CS340: Client/Server Development

Professor Jeff H. Sanford, M.S.

October 17, 2021

# CS 340: Project Two README

## About the Project/Project Title

This CRUD Python Module animal\_shelter2 allows for database access in adding documents to the Grazioso Salvare company’s Animals database and submitting inquiries. It validates processes along the way to ensure module is working correctly. It allows for editing and deleting of database entries as well if needed. Paired with the ProjectTwoDashboard jupyter file, it allows for the use of a dynamic browser-based dashboard to display and manipulate queries on the Animals database. The Dashboard displays the database data in a 10-unit spreadsheet style format data table with scrollable access and multiple page control. The Grazioso logo is displayed at the top left of the dashboard and allows access to their URL. The interactive data table is easily filtered through button control to display the following filtered results: Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking Rescue. It also allows for the data table to be reset. Selecting rows brings up a pie chart and geolocation map of selected animal below the data table. The Pie graph displays the distribution of outcome\_type: Adoption based on the breed of the animal. The geolocation map displays the location that the animal was rescued from.

**Motivation**

We were asked to develop this dashboard by the Grazioso Salvare Company to better access their database of rescued animals. By better being able to access and quickly pull up data from their database the better they will be able to find and place animals back in their proper home, or rehome animals as necessary.

## Getting Started

To begin with you are going to want to download MongoDB to your workstation. A link can be found below in the Installation section. Select the proper download for your system’s OS and install. You will utilize MongoDB through your local terminal/command prompt. You will also need a simple text editor like Sublime Text or some sort of Python IDE to create your python script. Finally, in order to test your functionality, you will need a program to test the functionality of your python method in correlation to your database. We utilized Jupyter Notebook on Linux based system.

**Steps Taken**

First, we created a CRUD module in Python that has methods that create, read, update, and delete database entries. This module allows us to also connect to the MongoDB datasets through authentication through PyMongo drivers. We then created a Dashboard with the help of Jupyter that would utilize our Python Module to access, display, and manipulate the data sets through a browser-based interface. The dashboard was created with callbacks allowing for the data sets to be displayed, filter, and return information to two interactive widgets. These widgets would include a pie chart and geolocation map.

## Installation

**MongoDB** - Go to [www.mongodb.com](http://www.mongodb.com) and hit the start free button. You’ll setup an account and have access to download MongoDB Atlas based on the workstation OS you are utilizing. MongoDB was chosen as the model component of the development because it offers a schema less design offering versatility to the company. It is also easy to scale out allowing the company to expand its database as needed. It pairs well with Python through PyMongo driver making manipulation easy and effective.

**Python IDE** – This is the software you will utilize to create your Python Script. Examples would include Visual Studio with Python IDE here: <https://visualstudio.microsoft.com/vs/features/python/>. Eclipse running PyDev IDE here: <https://www.eclipse.org/ide/>. Or a simple text editor like Sublime Text here: <https://www.sublimetext.com/>. We found that Sublime text was the best option as it is not cumbersome and easy to utilize. It is also freeware so it is cost effective.

**Test suite** – You can utilize a software like Jupyter Notebook: <https://jupyter.org/> to run tests on your python module with Mongodb. Or you can utilize a plugin like pytest: <https://pypi.org/project/pytest-mongodb/>.

## Usage

Here we highlight usage of the Python module and Dash framework through code examples and screenshots.

### Code Example

Here you can see a code example of the Python module where the module is broken down into five functions making up on AnimalShelter Class.

*from pymongo import MongoClient*

*from bson.objectid import ObjectId*

*from bson.json\_util import dumps*

*from bson import json\_util, 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:37211/AAC' % (username, password))*

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

**# Implements the C in CRUD.**

*def create(self, data):*

*if data is not None:*

*self.database.animals.insert(data)*

*return True*

*else:*

*return False*

**# Implements the R in CRUD.**

*def read(self, inquiry):*

*if inquiry is not None:*

*result = self.database.animals.find(inquiry,{"\_id":False})*

*return result*

*else:*

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

**# Implements the U in CRUD.**

*def update(self, inquiry, newValue):*

*if inquiry is not None:*

*result = self.database.animals.update\_one(inquiry, newValue)*

*doc = self.database.animals.find\_one(inquiry)*

*print(doc)*

*else:*

*raise Exception("Update not successful")*

**# Implements the D in CRUD.**

*def delete(self, inquiry):*

*if inquiry is not None:*

*result = self.database.animals.delete\_one(inquiry)*

*print(result)*

*for x in self.database.animals.find():*

*print(x)*

*else:*

*raise Exception("Delete not successful")*

Here you can see a code example of the Dashboard framework.

*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*

*import base64*

*from dash.dependencies import Input, Output, State*

*from dash import callback\_context*

*import json*

*from collections import OrderedDict*

*from bson.json\_util import dumps*

*import numpy as np*

*import pandas as pd*

*import matplotlib.pyplot as plt*

*from pymongo import MongoClient*

**#imports AnimalShelter Class from animal\_shelter2 Python module**

*from animal\_shelter2 import AnimalShelter*

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

**# Data Manipulation / Model**

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

**#username and password fed to AnimalShelter class Python module**

*username = "aacuser2"*

*password = "hello"*

*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')*

**#Adds in Grazioso Salvare’s logo**

*image\_filename = 'Grazioso Salvare Logo.png'*

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

*app.layout = html.Div([*

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

**#enodes Grazioso Salvare’s logo with URL href**

*html.A([*

*html.Img(*

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

*style={*

*'height' : '20%',*

*'width' : '20%',*

*'float' : 'left',*

*'position' : 'relative',*

*'padding-top' : 0,*

*'padding-right' : 0*

*})*

*], href='https://www.snhu.edu/'),*

*html.Br(),*

*html.Hr(),*

*html.Center(html.B(html.H1('Spencer Hayden: CS340 Dashboard'))),*

*html.Hr(),*

*html.Div(className='row',*

*style={'display' : 'flex'},*

*children=[*

*html.Button(id='button-one', n\_clicks=0, children='Water Rescue'),*

*html.Button(id='button-two', n\_clicks=0, children='Mountain Rescue'),*

*html.Button(id='button-three', n\_clicks=0, children='Disaster Rescue'),*

*html.Button(id='button-four', n\_clicks=0, children='Reset'),*

*]),*

*dash\_table.DataTable(*

*id='datatable-interactivity',*

*columns=[*

*{'id': i, 'name': i} for i in df.columns*

*],*

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

**#Makes table horizontally scrollable**

*style\_table={'overflowX': 'auto'},*

**#sets cell size**

*style\_cell={'height': 'auto', 'minWidth': '180px', 'width': '180px',*

*'maxWidth': 'auto', 'whiteSpace': 'normal'},*

*style\_cell\_conditional=([*

*{*

*'if':{'column\_id': i},*

*'textAlign': 'left'*

*} for i in ['animal\_type', 'name', 'breed', 'color',*

*'outcome\_subtype', 'outcome\_type',*

*'sex\_upon\_outcome']*

*]),*

*editable=False,*

*filter\_action="none",*

*sort\_action="native",*

*sort\_mode="multi",*

*column\_selectable=True,*

*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**

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

**# Filters interactive data table with MongoDB queries through buttons.**

*@app.callback(*

*Output('datatable-interactivity','data'),*

*[Input('button-one', 'n\_clicks'),*

*Input('button-two', 'n\_clicks'),*

*Input('button-three', 'n\_clicks'),*

*Input('button-four', 'n\_clicks')]*

*)*

*def update\_dashboard(btn1, btn2, btn3, btn4):*

*changed\_id = [p['prop\_id'] for p in callback\_context.triggered][0]*

*if 'button-one' in changed\_id:*

*df = pd.DataFrame(list(shelter.read({'$or': [{'breed':'Labrador Retriever'},*

*{'breed':'Chesapeake Bay Retriever'},*

*{'breed':'Newfoundland'}],*

*'age\_upon\_outcome\_in\_weeks':*

*{'$gte':26,'$lte':156},*

*'sex\_upon\_outcome':'Intact Female'})))*

*elif 'button-two' in changed\_id:*

*df = pd.DataFrame(list(shelter.read({'$or': [{'breed':'German Shepherd'},*

*{'breed':'Malamute'},*

*{'breed':'Old English Sheepdog'},*

*{'breed':'Siberian Husky'},*

*{'breed':'Rottweiler'}],*

*'age\_upon\_outcome\_in\_weeks':*

*{'$gte':26,'$lte':156},*

*'sex\_upon\_outcome':'Intact Male'})))*

*elif 'button-three' in changed\_id:*

*df = pd.DataFrame(list(shelter.read({'$or': [{'breed':'Doberman Pinscher'},*

*{'breed':'German Shepherd'},*

*{'breed':'Golden Retriever'},*

*{'breed':'Bloodhound'},*

*{'breed':'Rottweiler'}],*

*'age\_upon\_outcome\_in\_weeks':*

*{'$gte':20,'$lte':300},*

*'sex\_upon\_outcome':'Intact Male'})))*

*elif 'button-four' in changed\_id:*

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

*else:*

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

*return df.to\_dict('records')*

*@app.callback(*

*Output('datatable-interactivity', 'style\_data\_conditional'),*

*[Input('datatable-interactivity', 'selected\_columns')]*

*)*

*def update\_styles(selected\_columns):*

*return [{*

*'if': { 'column\_id': i },*

*'background\_color': '#D2F3FF'*

*} for i in selected\_columns]*

**#Updates Pie Chart based on row selection**

*@app.callback(*

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

*[Input('datatable-interactivity', "selected\_rows")],*

*[State('datatable-interactivity', "derived\_viewport\_data")]*

*)*

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

*if selected\_rows is None:*

*selected\_rows = []*

*if viewData is None:*

*dff = df*

*else:*

*dff = pd.DataFrame.from\_dict(selected\_rows)*

*return [*

*dcc.Graph(id='pie-chart',*

*figure = px.pie(*

*data\_frame=df,*

*values= {'outcome\_type': 'Adoption'},*

*names='breed',*

*color='breed',*

*))*

*]*

**#Updates Geolocation Map based on row selection**

*@app.callback(*

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

*[Input('datatable-interactivity', "selected\_rows")],*

*[State('datatable-interactivity', "derived\_viewport\_data")]*

*)*

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

*if selected\_rows is None:*

*selected\_rows = []*

*if viewData is None:*

*dff = df*

*else:*

*dff = pd.DataFrame.from\_dict(selected\_rows)*

*return [*

*dl.Map(style={'width': '1000px', 'height': '500px'}, center=[dff.iloc[0,13],dff.iloc[0,14]], zoom=10, children=[*

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

**# Marker with tool tip and popup**

*dl.Marker(position=[dff.iloc[0,13],dff.iloc[0,14]], children=[*

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

*dl.Popup([*

*html.H1(list(shelter.read({'animal\_id'}))),*

*html.P(dff.iloc[0,9])*

*])*

*])*

*])*

*]*

*app*

### Dash framework: Screenshots

*Graphical user interface, text, application, email

Description automatically generated*

*Example of importing libraries needed for implementation of Dashboard.*

*Graphical user interface, text, application, email

Description automatically generated*

*Graphical user interface, text, application

Description automatically generated*

*Example of creating an authenticated user within the Mongo Shell through connecting Dash to Python Module initialize database def.*

*Text

Description automatically generated*

*Example of adding in Company Logo linked to URL through href.*

*Graphical user interface, text

Description automatically generated*

*Example of the creation of the filtration buttons in the start of the creation of the dash data table.*

*Graphical user interface, text, application

Description automatically generated*

*Example of further creating dash datatable including dash graph and map widgets.*

*Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated*

*Example of creating filtered returns tied to buttons already created. 4 buttons each filtering data table results when buttons are pressed.*

*Text

Description automatically generated*

*Example of pie chart widget updated through row selection.*

*Graphical user interface, text, application, email

Description automatically generated*

*Example of geolocation map widget updated through row selection.*

*Graphical user interface, table

Description automatically generated*

## *Example of resulting dashboard.*

**Challenges**

## The main challenges that were faced were with the dash libraries being integrated into the project properly. Specifically, the libraries that allow for button callback\_context to be utilized properly. Numerous hours were spent trying to figure out the issues. Both user documentation: <https://dash-docs.herokuapp.com/dash-html-components/button> and other resources were of no help in solving this problem.

## Roadmap/Features (Optional)

Further development is to include database updating and delete features through the dashboard. This will allow for further control of the editing and management of the database. Filtration errors to be debugged and fixed in next release.

## Contact

Your name: Spencer Hayden