Module Seven Project Two

Anthony Lee

SNHU

CS-340-T5628 – Client/Server Development

Professor Tad Kellogg

June 19, 2022

# CS 340 README for Module Four Milestone

## About the Project/Project Title

The purpose of this project is to get students used to developing with MongoDB to create a database that can interact with client-side code.

## Motivation

This project was to demonstrate the use of a dashboard using Mongo and Python.

Mongo was used because it provides a quick setup of the database from a csv document with a Python-friendly interface.

The purpose of the CRUD Python module is they are the basic methods of interacting with a MongoDB server. CRUD stands for Create, Read, Update, and Delete. These Crud methods are the primary ways to manage data in a database.

The dashboard displays a table, pie graph, and geolocation of a list of animals given by a document.

## Getting Started

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

1. In a Mongo shell, upload the “aac\_shelter\_outcomes.csv” using the appropriate MongoDB import tool.
2. Using a user with Admin permissions, create a user called aacuser and give it read/write access to AAC database. Example:

Text

Description automatically generated

1. Exit the mongo shell, disconnect from the mongo shell, and reconnect as aacuser
2. Develop a CRUD class that, when instantiated, provides the following functionality:
   1. A method that inserts a document into a specified MongoDB database and collection
      1. Input: argument to function will be set of key/value pairs in the data type acceptable to the MongoDB driver insert API call
      2. Return: “True” if successful insert, else “False”
      3. Example:

# 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")  
 return False

* 1. A method that queries for documents from a specified MongoDB database and specified collection
     1. Input: arguments to function should be the key/value look pair to use with MongoDB driver find API call
     2. Return: result in cursor if successful, else MongoDB returned error message
     3. Example:

# Create method to implement the R in CRUD.   
 def read(self,criteria):  
 # criteria is not None then this find will return all rows which matches the criteria  
 if criteria is not None:  
 #criteria should be a dictionary  
 return self.database.animals.find(criteria, {"\_id": False})  
   
 else:  
 #if there is no search criteria, then error   
 raise Exception("Nothing to read, because data parameter is empty")  
 return False

* 1. A method that queries for and changes document(s) from a specified MongoDb database and specified collection
     1. Input: arguments to function should be the key/value look pair to use with MongoDB driver find API call
     2. Return: result in JSON format if successful, else MongoDB returned error message
     3. Example:

# Create method to implement the U in CRUD.  
 def update(self, fromTarget, toTarget):  
 if fromTarget is not None:  
 #update   
 update\_result = self.database.animals.update\_many(fromTarget, toTarget)  
 print(update\_result)  
   
   
 else:  
 raise Exception("Error")  
 return False

* 1. A method that queries for and removes document(s) from a specified MongoDB database and specified collection
     1. Input: arguments to function should be the key/value look pair to use with MongoDB driver find API call
     2. Return: result in JSON format if successful, else MongoDB returned error message

Example: # Create method to implement the D in CRUD.  
 def delete(self, criteria):  
 if criteria is not None:  
 #delete  
 try:  
 delete\_result = self.database.animals.delete\_many(criteria)  
 print(delete\_result)  
 return True  
   
 except Exception as e:  
 print("An exception has occurred: ", e)  
 return False  
   
 else:  
 # lets user know there was a problem  
 raise Exception("Nothing to delete, because the criteria parameter is empty")  
 return False

1. Test using Jupyter
   1. Example:

Graphical user interface, application

Description automatically generated

1. Set up Dash framework
   1. Example: Graphical user interface, text, application

      Description automatically generated
2. Create layout
   1. Example:

Text

Description automatically generated

1. Create a filter to be able to dynamically update dashboard while in use
   1. Example for filtering type of breeds:

Text, timeline

Description automatically generated

1. Add a pie chart that can take filter and dynamically update
   1. Example:

Text

Description automatically generated

1. Add a geolocation that can take the filtered data and dynamically update while displaying key information about the animal selected:
   1. Example:

Text

Description automatically generated

## Installation

Tools needed:

**Python (3.x.x):** https://www.python.org/downloads/windows/

**Jupyter Notebook**: https://jupyter.org/try

**PyMongo**: https://pypi.org/project/pymongo/

MongoDB: https://www.mongodb.com/try/download/community

**Plotyly**: https://www.journaldev.com/19692/python-plotly-tutorial#:~:text=Installation.%20To%20install%20plotly%2C%20open%20a%20terminal%20window,to%20install%20to%20collect%20dependencies%20and%20download%20them%3A

**Plotly** must be imported in order to generate the proper charts. Plotly is a charting tool for Python applications.

**Dash**: https://pypi.org/project/dash/

Dash is a framework used to build web applications. Dash Core Components can be imported to Jupyter Notebook.

**Pandas**: https://pandas.pydata.org/pandas-docs/stable/getting\_started/install.html

Pandas is a tool for Python that creates the data frames.

## Usage

This project shows the basics of the CRUD methods and demonstrates how they are used. They are Create, Read, Update, and Delete. Read is also implements them for use in a dashboard to display data given through a Mongo database.

There are currently 3 main functions. The first uses radio buttons to sort data based on rescue dog type desired by Grazioso Salvare. Clicking on a specific radio button runs the database queries and returns an updated data frame. The reset radio button resets the results to an unfiltered state. The second function is a geolocation map that dynamically updates based on the radio button clicked for a specific row, which is for a specific dog. The map will then show the geolocation of the dog with information about sex, age, name, and breed. The third function is the dynamic pie chart. It sorts the information that is displayed currently on the table and dynamically updates when the table is changed. It is sorted by breed. Since the table only shows 10 entries at a time, the pie chart does not display all the data, only the 10 displayed.

### Code Example

CRUD:

from pymongo import MongoClient  
from bson.objectid import ObjectId  
  
class AnimalShelter(object):  
 *""" CRUD operations for Animal collection in MongoDB """* def \_\_init\_\_(self, username=None, password=None):  
 # Initializing the MongoClient. This helps to   
 # access the MongoDB databases and collections.  
 if username and password:  
 self.client = MongoClient('mongodb://%s:%s@localhost:54698' % (aacuser, password))  
 else:  
 self.client = MongoClient('mongodb://localhost:54698')  
   
 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")  
 return False  
  
# Create method to implement the R in CRUD.   
 def read(self,criteria):  
 # criteria is not None then this find will return all rows which matches the criteria  
 if criteria is not None:  
 #criteria should be a dictionary  
 return self.database.animals.find(criteria, {"\_id": False})  
   
 else:  
 #if there is no search criteria, then error   
 raise Exception("Nothing to read, because data parameter is empty")  
 return False  
   
# Create method to implement the U in CRUD.  
 def update(self, fromTarget, toTarget):  
 if fromTarget is not None:  
 #update   
 update\_result = self.database.animals.update\_many(fromTarget, toTarget)  
 print(update\_result)  
   
   
 else:  
 raise Exception("Error")  
 return False  
   
# Create method to implement the D in CRUD.  
 def delete(self, criteria):  
 if criteria is not None:  
 #delete  
 try:  
 delete\_result = self.database.animals.delete\_many(criteria)  
 print(delete\_result)  
 return True  
   
 except Exception as e:  
 print("An exception has occurred: ", e)  
 return False  
   
 else:  
 # lets user know there was a problem  
 raise Exception("Nothing to delete, because the criteria parameter is empty")  
 return False

Accessing database for dashboard:

# change animal\_shelter and AnimalShelter to match your CRUD Python module file name and class name  
from AnimalShelter import AnimalShelter  
  
###########################  
# Data Manipulation / Model  
###########################  
# FIX ME update with your username and password and CRUD Python module name  
  
username = "aacuser"  
password = "password"  
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:

#########################  
# Dashboard Layout / View  
#########################  
app = JupyterDash('SimpleExample')  
  
# 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.Center(html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()))),  
 html.Center(html.B(html.H1('SNHU CS-340 Dashboard Anthony Lee'))),  
 html.Hr(),  
 html.Div(  
  
 # *FIXME Add in code for the interactive filtering options. For example, Radio buttons, drop down, checkboxes, etc.* # Radio Items to select the rescue filter options  
 dcc.RadioItems(  
 id='filter-type',  
 # created the labels and keys based on the Grazioso requirements  
 options=[  
 {'label': 'Water Rescue', 'value': 'WR'},  
 {'label': 'Mountain/Wilderness Rescue', 'value': 'MWR'},  
 {'label': 'Disaster Rescue/Individual Tracking', 'value': 'DRIT'},  
 {'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'),  
 # *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  
 editable=False,  
 filter\_action="native",  
 sort\_action="native",  
 sort\_mode="multi",  
 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',  
 )  
 ])  
])

Controllers to update dashboard:

@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  
 # adjusts the read request  
 if filter\_type == 'WR':  
 df = pd.DataFrame(list(shelter.read({'$and': [{'sex\_upon\_outcome': 'Intact Female'},  
 {'$or': [  
 {'breed': 'Labrador Retriever Mix'},  
 {'breed': 'Chesapeake Bay Retriever'},  
 {'breed': 'Newfoundland'}]  
 },  
 {'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 26}},  
 {'age\_upon\_outcome\_in\_weeks': {'$lte': 156}}]  
 }]  
 })))  
 # adjusts the read request  
 elif filter\_type == 'MWR':  
 # breeds and ages determined by Grazioso  
 df = pd.DataFrame(list(shelter.read({'$and': [{'sex\_upon\_outcome': 'Intact Male'},  
 {'$or': [  
 {'breed': 'German Shepherd'},  
 {'breed': 'Alaskan Malamute'},  
 {'breed': 'Old English Sheepdog'},  
 {'breed': 'Rottweiler'},  
 {'breed': 'Siberian Husky'}]  
 },  
 {'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 26}},  
 {'age\_upon\_outcome\_in\_weeks': {'$lte': 156}}]  
 }]  
 })))  
 # adjusts the read request  
 elif filter\_type == 'DRIT':  
 # breeds and ages determined by Grazioso  
 df = pd.DataFrame(list(shelter.read({'$and': [{'sex\_upon\_outcome': 'Intact Male'},  
 {'$or': [  
 {'breed': 'Doberman Pinscher'},  
 {'breed': 'German Shepherd'},  
 {'breed': 'Golden Retriever'},  
 {'breed': 'Bloodhound'},  
 {'breed': 'Rottweiler'}]  
 },  
 {'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 20}},  
 {'age\_upon\_outcome\_in\_weeks': {'$lte': 300}}]  
 }]  
 })))  
 # resets the search to nothing to allow all results to be displayed  
 elif filter\_type == 'RESET':  
 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)

Controller to update graph:

@app.callback(  
 Output('graph-id', "children"),  
 [Input('datatable-id', "derived\_viewport\_data")])  
def update\_graphs(viewData):  
 ###FIX ME ####  
 # import the current displayed data  
 dff = pd.DataFrame.from\_dict(viewData)  
 # create the values needed for breed type and count  
 names = dff['breed'].value\_counts().keys().tolist()  
 values = dff['breed'].value\_counts().tolist()  
 # create pie chart based on data  
 return [  
 dcc.Graph(  
 figure=px.pie(  
 data\_frame=dff,  
 values=values,  
 names=names,  
 color\_discrete\_sequence=px.colors.sequential.RdBu,  
 width=800,  
 height=500  
 )  
 )  
 ]

Controller to update map:

@app.callback(  
 Output('map-id', "children"),  
 [Input('datatable-id', "derived\_viewport\_data"),  
 Input('datatable-id', 'selected\_rows'),  
 Input('datatable-id', 'selected\_columns')])  
def update\_map(viewData, selected\_rows, selected\_columns):  
 # *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)  
 if selected\_rows == []:  
 selected\_rows = [0]  
 # 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=[(dff.iloc[selected\_rows[0], 13]), (dff.iloc[selected\_rows[0], 14])], children=[  
 dl.Tooltip(dff.iloc[selected\_rows[0], 4]),  
 dl.Popup([  
 html.H4("Animal Name"),  
 html.P(dff.iloc[selected\_rows[0], 9]),  
 html.H4("Sex"),  
 html.P(dff.iloc[selected\_rows[0], 12]),  
 html.H4("Breed"),  
 html.P(dff.iloc[selected\_rows[0], 4]),  
 html.H4("Age"),  
 html.P(dff.iloc[selected\_rows[0], 15])  
 ])  
 ])  
 ])  
 ]

### Screenshots

Import

A screenshot of a computer

Description automatically generated

Administrator account

Text

Description automatically generated

aacuser account

Graphical user interface, text

Description automatically generated

Dashboard

Graphical user interface, text

Description automatically generated with medium confidence

Water Rescue

Text

Description automatically generated

Mountain or Wilderness Rescue

Text

Description automatically generated with medium confidence

Disaster or Individual Tracking

Text

Description automatically generated with medium confidence

Reset

Graphical user interface, text

Description automatically generated

Test CRUD:

Graphical user interface, application

Description automatically generated

## Challenges Encountered:

The biggest challenge that I encountered was getting the map to dynamically update when a new filter option was chosen. It was not until I got feedback for the last assignment that I was able to figure out how to implement it. Another issue I encountered was my previous iteration of the dash table let me select multiple options, which ran into an issue with the map trying to display multiple animals at once. To fix this, I changed the code to only allow a single choice to be selected.

## Contact

Your name: Anthony Lee

anthony.lee6@snhu.edu