Project 2 README

Cs-340

James Carver

2023

**Rescue Animal Application**

**About the project**

We were given the job of creating the most recent application for Grazioso Salvare, a business that chooses dogs for search and rescue training, as a part of Global Rain, a recognized software firm. To identify and classify dogs, this research employs pre-existing data from numerous animal shelters. This application was created as a full-stack solution for the business, meeting all of their data handling requirements and offering a crucial front-end interface that the staff could easily utilize. We were given a database but wanted the best ways to handle and manipulate the data. This project makes use of the Model-View-Controller (MVC) software architecture, which enables us to keep the data model's simplicity of operation. We start manipulating the data in the application's backend using Mongo Database. Dash, an open-source Python framework for making rapid analytical programs, is used to build our front-end system. Last but not least, we link Mongo and Dash using a unique Python module that offers easy utilization of capabilities to get the information we require and return it to the user as soon as possible.

**Motivation**

Our backend, which has direct data access, is powered by the quick and effective NoSQL database program MongoDB. This program utilizes JSON-based formatting to store documents. Using our data-driven approach, Dash, an open-source Python framework, enables the optimization of a front-end interface. Dash is widely used for analytical web applications. Python is used to connect everything, making dynamic typing simple to use and providing a quick approach to connecting various parts of a web application.

**Functionality**

This is the “Home Page”, or the first screen you see. Data is imported in no particular order. The columns and rows are imported from data in MongoDB. There are 4 predetermined radio buttons to filter per the customer’s request. All columns are sortable and have a text filter. Below is a pie chart with the number of available animals and a map of the location of the chosen animal. To the left, rows are selectable to show on the map. The image is a clickable URL to the home page.

A picture containing graphical user interface

Description automatically generated

The next 4 slides are examples of each radio filter button in action.

A picture containing diagram

Description automatically generatedGraphical user interface, application

Description automatically generated

A picture containing application

Description automatically generatedA picture containing graphical user interface

Description automatically generated

**Tools**

MongoDB - [*https://www.mongodb.com/docs/manual/installation/*](https://www.mongodb.com/docs/manual/installation/)

[*https://www.mongodb.com/docs/v4.2/introduction/*](https://www.mongodb.com/docs/v4.2/introduction/)

Python - [*https://www.python.org/downloads/*](https://www.python.org/downloads/)

Dash Framework - [*https://dash.plotly.com/installation*](https://dash.plotly.com/installation)

[*https://dash.plotly.com/introduction*](https://dash.plotly.com/introduction)

Jupyter - [*https://jupyter.org/install*](https://jupyter.org/install)

MongoDB & Python

“Python, the top programming language for data science, and MongoDB, with its flexible and dynamic schema, are a fantastic match for building modern web applications, JSON APIs, and data processors, just to name a few. MongoDB has a native Python driver and a team of engineers dedicated to making sure MongoDB and Python work together flawlessly. PyMongo, the standard MongoDB driver library for Python, is easy to use and offers an intuitive API for accessing databases, collections, and documents” (MongoDB, n.d.).

MongoDB is an open-source document database that employs a flexible schema for data storage and is designed on a horizontal scale-out architecture. Each record in a MongoDB database is a document specified in BSON, a binary representation of the data, as opposed to being stored in rows and columns like SQL databases. Programs can then retrieve this information in a JSON format. Scale-out architecture, which enables numerous small machines to collaborate and build quick systems and manage massive amounts of data, is the foundation upon which MongoDB was created. This makes it a great match for Python. Many different libraries are available in Python to work with large data sets, and is much faster than other programming languages.

Dash Framework

The open-source Python package Dash is used to build responsive web applications. Building and delivering data apps with unique user interfaces is perfect for Dash. It is especially appropriate for anyone who handles data. This enables us to make the tables and widgets that users will use to view, sort, and filter our data.

**Getting Started**

*To get a local copy up and running, follow these simple example steps. Please download and install the latest version of Python and MongoDB. For this project, I used Jupyter Notebook. You can use any IDE.* [*https://www.mongodb.com/docs/manual/installation/*](https://www.mongodb.com/docs/manual/installation/)*<https://www.python.org/downloads/>*

1. *The first step is to start MongoDB in the Linux shell.* **mongod\_clt start-noauth***. Some systems might be sudo systemctl start mongodb.*
2. *Then, using* **mongoimport***, import the* **aac\_shelter\_outcome.csv** *file into a database called* **AAC** *and using* **animals** *as a collection name using your local port. Help -* [*https://docs.mongodb.com/manual/reference/program/mongoimport/*](https://docs.mongodb.com/manual/reference/program/mongoimport/)
3. *Start mongo by typing “***mongo”** *and hitting enter. Verify file was imported into the AAC database by typing “***show dbs”** *then* **“use AAC”** *then “***show collections”. Do not include the “ ”.**

*Graphical user interface, text

Description automatically generated*

1. *After verifying, create a secure login for the database.*

[*https://www.mongodb.com/docs/manual/tutorial/create-users/*](https://www.mongodb.com/docs/manual/tutorial/create-users/)

A screenshot of a computer

Description automatically generated

1. *Create a python class to create, read, update, and delete (CRUD) files in the database. For this I used* **Mastering MongoDB 4.x** *by* **Alex Giamas***. The hardest part of this step was getting the authentication to connect to the database. For testing purposes, I hardcoded the username and password into the python code.*

from pymongo import MongoClient

from bson.objectid import ObjectId

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:29948/?authMechanism=DEFAULT&authSource=AAC' % (username, password))

# where xxxx is your unique port number

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

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

def create(self, data): # TODO add check for existing entry

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")

# Create method to implement the R in CRUD.

def read\_all(self, data):

cursor = self.database.animals.find(data, {'\_id':False}) ## return a cursor which a pointer to a list of results

return cursor

def read(self, data):

return self.database.animals.find\_one(data) ## returns only one document as a python dictionary

# Create method to implement the U in CRUD.

def update\_many(self, query, update):

if query is not None:

self.database.animals.update\_many(query, update)

return True

else:

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

def update\_one(self, query, update):

if query is not None:

self.database.animals.update\_one(query, update)

return True

else:

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

# Create method to implement the D in CRUD

def delete(self, data):

if data is not None:

self.database.animals.remove(data)

return True

else:

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

1. *Create the main dashboard for the client side. This is where the data is displayed and manipulated. This is also where you can get creative with the design of the dashboard as long as it still serves the requirements.*

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

import base64

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

from animal\_shelter import AnimalShelter

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

# Data Manipulation / Model

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

# username and password hardcoded for testing

username = "aacuser"

password = "jvcarver"

shelter = AnimalShelter(username, password)

# class read method must support return of cursor object

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

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

# Dashboard Layout / View

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

app = JupyterDash('Final Project')

#FIXME

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'}),

html.A([

html.Img(

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

style = {

'width':'10%',

'height':'10%'

})

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

html.Center(html.B(html.H1('Created by: James Carver'))),

html.Hr(),

html.Div(

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

dcc.RadioItems(

id='filter-type',

options=[

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

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

{'label': 'Disaster Rescue', 'value': 'Disaster Rescue'},

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

],

value='Reset',

)

),

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'),

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,

style\_cell={'textAlign': 'left','padding': '5px'},

style\_header={

'backgroundColor': 'rgb(210, 210, 210)',

'color': 'black',

'fontWeight': 'bold'

},

style\_data\_conditional=[

{

'if': {'row\_index': 'odd'},

'backgroundColor': 'rgb(220, 220, 220)',

}

],

),

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('filter-type', 'value')])

def update\_dashboard(filter\_type):

if filter\_type == 'Water Rescue':

#Grazioso breeds and age requirement

df = pd.DataFrame(list(shelter.read\_all({'$and': [{'sex\_upon\_outcome': 'Intact Female'},

{'$or': [

{'breed': 'Labrador Retriever Mix'},

{'breed': 'Chesa Bay Retr Mix'},

{'breed': 'Newfoundland Mix'},

{'breed': 'Newfoundland/Labrador Retriever'},

{'breed': 'Newfoundland/Australian Cattle Dog'},

{'breed': 'Newfoundland/Great Pyrenees'}]

},

{'$and': [{'age\_upon\_outcome\_in\_weeks': {'$gte': 26}},

{'age\_upon\_outcome\_in\_weeks': {'$lte': 156}}]

}]

})))

#adjusts the read request for the desired dog type and status

elif filter\_type == 'Mountain Rescue':

#Grazioso breeds and age requirement

df = pd.DataFrame(list(shelter.read\_all({'$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 for the desired dog type and status

elif filter\_type == 'Disaster Rescue':

#Grazioso breeds and age requirement

df = pd.DataFrame(list(shelter.read\_all({'$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 no filter

elif filter\_type == 'Reset':

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

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]

#update the pie chart

@app.callback(

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

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

def update\_graphs(viewData):

###FIX ME ####

# imports the currently displayed data

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

#creates the values needed breeds and counts

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

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

#creates a pie chart based on the data above

return [

dcc.Graph(

figure = px.pie(

data\_frame=dff,

values = values,

names = names,

color\_discrete\_sequence=px.colors.sequential.RdBu,

width=800,

height=500

)

)

]

@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):

# imports the currently viewed data

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

#determines the default status

if selected\_rows == []:

selected\_rows = [0]

# creats a map for a single selected row or the default

if len(selected\_rows) == 1:

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])

])

])

])

]

App

**Challenges**

During this project I only encountered two challenges. The first was the connection between Python and MongoDB. First, the authentication on MongoDB must be set up correctly. In this step, it took me many tries because the syntax must be perfect. Also, the coding in Python must be specific. Example, self.client = MongoClient('mongodb://%s:%s@localhost:29948/?authMechanism=DEFAULT&authSource=AAC' % (username, password)), the localhost must match your port number. I would suggest doing all coding without using authentication first for testing. The second was deciding what to use on the dashboard. I ended up with a basic functional design.

MongoDB. (n.d.). *Build A Python Database With*. https://www.mongodb.com/languages/python