# CS 340 README

## About the Project/Project Title

**AnimalShelter CRUD Module**  
This project is a Python-based CRUD module for managing animal records in a MongoDB database. It provides functionalities to create, read, update, and delete records in a structured and efficient manner. This project includes an interactive dashboard for the client, Grazioso Salvare, allowing them to interact with and visualize the database effectively.

## Motivation

The project aims to facilitate seamless interaction with animal shelter data, allowing staff to manage records efficiently and accurately. The motivation is to enhance operational efficiency by minimizing manual data handling and providing a user-friendly interface for database operations. Grazioso Salvare needs to identify suitable dogs for search-and-rescue training based on specific criteria, and this project addresses that need.

**Functionality of the Project**

The functionality of the project includes:

* Providing a user-friendly interface to manage animal shelter records.
* Allowing users to create, read, update, and delete records in the MongoDB database.
* Offering interactive filtering options to categorize dogs based on rescue suitability.
* Displaying an interactive data table that responds to filter selections.
* Featuring a geolocation chart and an additional chart (age distribution pie chart) that dynamically update based on filtered data.

## Getting Started

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

1. **Install MongoDB**: Follow the [official guide](https://docs.mongodb.com/manual/installation/).
2. **Install Python**: Download and install from [python.org](https://www.python.org/).
3. **IDE**: User chosen IDE I used Jupyter Notebook official [installation guide.](https://jupyter.org/install)
4. **Install Required Python Packages**:

pip install pymongo

pip install dash

pip install jupyter-dash

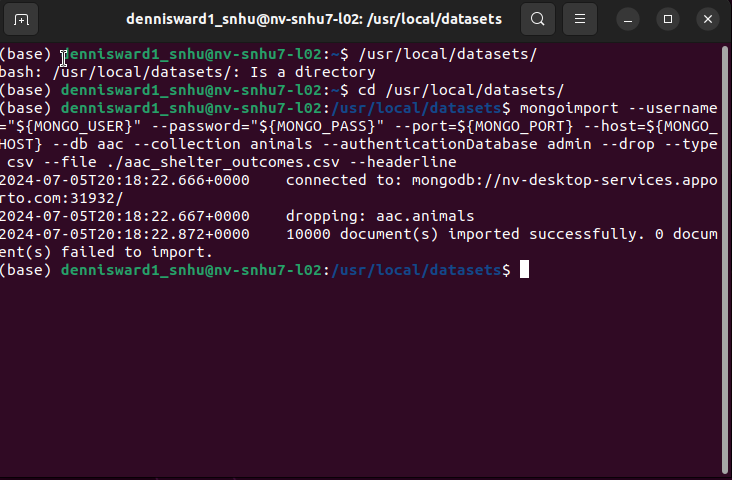
pip install dash-leaflet

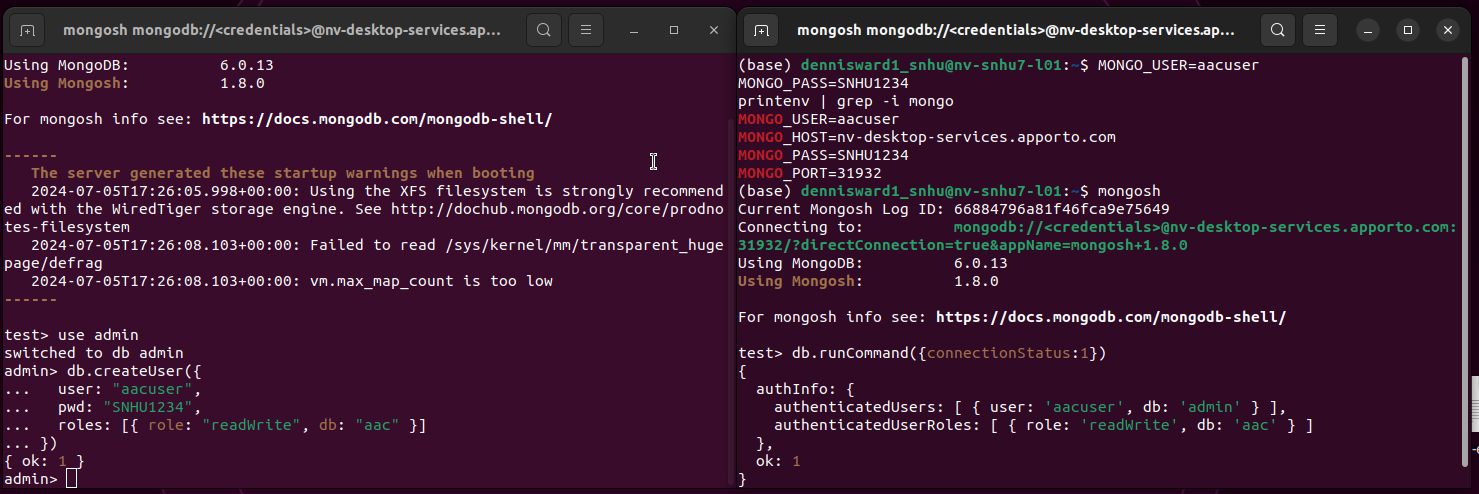
pip install plotly

pip install pandas

1. **Download the Dataset:** Obtain the Austin Animal Center Outcomes dataset from the Supporting Materials section.
2. **Import the Dataset:**  below is the command used to import the dataset into our database

mongoimport --username="${MONGO\_USER}" --password="${MONGO\_PASS}" --port=${MONGO\_PORT} --host=${MONGO\_HOST} --db aac --collection animals --authenticationDatabase admin --drop --type csv --file ./aac\_shelter\_outcomes.csv --headerline

1. **Create a User Account :**  User “aacuser” was created(left), connection verified(right bottom), and connection information obtained below(right top) .



**Create User Command:**

mongosh

use admin

db.createUser({

  user: "aacuser",

  pwd: "SNHU1234",

  roles: [{ role: "readWrite", db: "aac" }]

})

**Login to User:**

Open new Terminal and verify Login credentials

MONGO\_USER=aacuser

MONGO\_PASS=SNHU1234

printenv | grep -i mongo

mongosh

db.runCommand({connectionStatus:1})

**Database and User Authentication Setup**:

* The MongoDB database was set up with a user aacuser and password SNHU1234 for authenticated access.
* The database and collection were specified (aac database and animals collection).
* Connection information checked and verified.
* AnimalShelter(username, password) initialized with Username, Password to verify client side connection.

**Creation of CRUD Operations**

1. **Create (C):**

* Implemented using insert\_one to add new records.
* Input: Dictionary representing the record.
* Returns: True if successful, otherwise False.

1. **Read (R):**

* Implemented using find to query records.
* Input: Dictionary with query parameters or None to return all.
* Returns: List of matching documents.

1. **Update (U):**

* Implemented using update\_many to modify existing records based on a query.
* Inputs: Dictionary with query parameters and a dictionary with update data.
* Returns: Number of documents modified.
* Ensured only non-empty update data was processed to avoid errors.

1. **Delete (D):**

* Implemented using delete\_many to remove records based on a query.
* Input: Dictionary with query parameters.
* Returns: Number of documents deleted.
* Added validation to prevent accidental deletions by ensuring the query parameter was not empty.

## Installation

**Tools Used:**

1. **MongoDB:** Chosen for its flexibility, scalability, and ease of use with unstructured data.
2. **Python:** Selected for its readability and extensive library support.
3. **pymongo:** MongoDB driver for Python, providing a simple interface for database operations.
4. **Dash:** Used to create the interactive web application dashboard.

**Rationale for Using MongoDB:**

* **Flexibility:** Handles unstructured data efficiently.
* **Scalability:** Easily scales with growing data needs.
* **Integration:** Seamlessly integrates with Python via pymongo.

**Rationale for Using Dash:**

* **Interactive Visualization:** Provides dynamic and interactive data visualization capabilities.
* **Ease of Use:** Simplifies the creation of web applications with minimal code.
* **Community Support:** Extensive documentation and community resources.

**Installation Steps**:

1. Install MongoDB from the [official guide](https://docs.mongodb.com/manual/installation/).
2. Install Python from the [python.org](https://www.python.org/).
3. Install pymongo using the command pip install pymongo.

## Usage

Here are some examples of how the MongoDB CRUD implementaion works and how it can be used:

### Code Example

This is a simple implementation of how you can use the code.

# Instantiate AnimalShelter

USER = 'aacuser'

PASS = 'SNHU1234'

shelter = animal\_shelter.AnimalShelter(USER, PASS)

# Creating a new record

shelter.create({"name": "Milo", "age": 3, "species": "Cat"})

# Reading records

cats = shelter.read({"species": "Cat"})

print(cats)

# Updating a record

shelter.update({"name": "Milo"}, {"age": 4})

# Deleting a record

shelter.delete({"name": "Milo"})

### Tests

You should be able to run this test case to ensure proper implementation.

from animal\_shelter import AnimalShelter

import animal\_shelter

from pymongo import MongoClient

import animal\_shelter

# Instantiate AnimalShelter

USER = username ='aacuser'

PASS = password ='SNHU1234'

shelter = animal\_shelter.AnimalShelter(username,password)

# create

data = {"animal\_id": "A123456", "name": "Jack", "type": "Dog"}

create\_result = shelter.create(data)

print(f"Create operation successful: {create\_result}")

#read

query = {"type": "Dog"}

read\_result = shelter.read(query)

print("Read result: ")

for doc in read\_result:

    print(doc)

#update

update\_query = {"animal\_id": "A123456"}

update\_data = {"name": "Jack Updated"}

update\_result = shelter.update(update\_query, update\_data)

print(f"Update operation successful: {update\_result}")

#delete

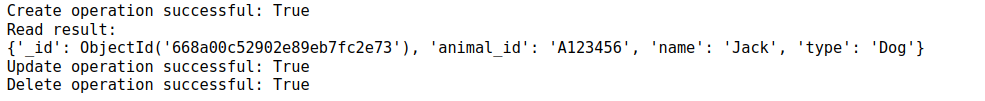
delete\_query = {"animal\_id": "A123456"}

delete\_result = shelter.delete(delete\_query)

print(f"Delete operation successful: {delete\_result}")

### Screenshots

The output from the Test case above should be:



The screenshot of my own run of the test case in Jupyter Notebook:



**Advanced Usage**

Creation of Dashboard using Dash

Install needed modules if you have not yet done so:

pip install pymongo

pip install dash

pip install jupyter-dash

pip install dash-leaflet

pip install plotly

pip install pandas

pip install matplotlib

**Imports:**

# Setup the Jupyter version of Dash

from jupyter\_dash import JupyterDash

# Configure the necessary Python module imports for dashboard components

import dash\_leaflet as dl

from dash import dcc

from dash import html

import plotly.express as px

from dash import dash\_table

from dash.dependencies import Input, Output, State

import base64

# Configure the plotting routines

import pandas as pd

# Import the CRUD module

from animal\_shelter import AnimalShelter

**Data Manipulation/Model:**

# Update with your username and password and CRUD Python module name

username = "aacuser"

password = "SNHU1234"

# Connect to database via CRUD Module

db = AnimalShelter(username, password)

# Read data from the database

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

# Remove the '\_id' column to prevent DataTable crash

df.drop(columns=['\_id'], inplace=True)

# Add in Grazioso Salvare’s logo

image\_filename = 'Grazioso Salvare Logo.png'  # Replace with your own image path

encoded\_image = None

try:

    with open(image\_filename, 'rb') as image\_file:

        encoded\_image = base64.b64encode(image\_file.read()).decode()

except FileNotFoundError:

    print(f"File {image\_filename} not found.")

**Initialize and Configure layout:**

#initialize JupyterDash instance

app = JupyterDash(\_\_name\_\_)

#Layout of Dashboard

app.layout = html.Div([

    #Image and link embeded.

    html.Center(html.A(href='https://www.snhu.edu', children=[

        html.Img(src=f'data:image/png;base64,{encoded\_image}', style={'height': '100px'})

    ])),

    #Title

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

    html.Hr(),

    html.Div([

        # Radio buttons for selecting rescue type

        dcc.RadioItems(

            id='rescue-type-radio',

            options=[

                {'label': 'Water Rescue', 'value': 'water\_rescue'},

                {'label': 'Mountain or Wilderness Rescue', 'value': 'mountain\_rescue'},

                {'label': 'Disaster or Individual Tracking', 'value': 'disaster\_tracking'},

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

            ],

            value='reset',  # Default filter

            labelStyle={'display': 'block'}  # Display each radio item on a new line

        ),

    ]),

    html.Hr(),

    #unique Identifier

    html.Div(id="unique-id", children="Dennis Ward"),

    #Data Table

    dash\_table.DataTable(

        id='datatable-id',

        columns=[{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns],

        data=df.to\_dict('records'),

        #Filtering and sorting options.

        editable=True,

        filter\_action="native",

        sort\_action="native",

        sort\_mode="multi",

        row\_selectable="single",

        selected\_rows=[],

        page\_action="native",

        page\_current=0,

        page\_size=10,

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

        style\_cell={

            'minWidth': '150px', 'width': '150px', 'maxWidth': '150px',

            'whiteSpace': 'normal',

            'textAlign': 'left',

        },

    ),

    html.Br(),

    html.Hr(),

    #Graph and Map widgets

    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',

            )

        ]

    )

])

**Radio Button Callback Controller:**

# Callback to update the DataTable based on the selected rescue type from radio buttons

@app.callback(Output('datatable-id', 'data'),

              [Input('rescue-type-radio', 'value')])

def update\_dashboard(rescue\_type):

    # Define the queries for each rescue type

    queries = {

        'water\_rescue': {

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

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

            'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}

        },

        'mountain\_rescue': {

            'breed': {'$in': ['German Shepherd', 'Alaskan Malamute', 'Old English Sheepdog', 'Siberian Husky', 'Rottweiler']},

            'sex\_upon\_outcome': 'Intact Male',

            'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}

        },

        'disaster\_tracking': {

            'breed': {'$in': ['Doberman Pinscher', 'German Shepherd', 'Golden Retriever', 'Bloodhound', 'Rottweiler']},

            'sex\_upon\_outcome': 'Intact Male',

            'age\_upon\_outcome\_in\_weeks': {'$gte': 20, '$lte': 300}

        },

        'reset': {}  # Query to reset the filter

    }

    # Select the appropriate query based on the selected rescue type

    query = queries[rescue\_type]

    # Retrieve the filtered data from the database

    filtered\_data = pd.DataFrame.from\_records(db.read(query))

    # Remove the '\_id' column to prevent DataTable crash

    filtered\_data.drop(columns=['\_id'], inplace=True)

    # Fill missing values with "Undocumented"

    filtered\_data.fillna("Undocumented", inplace=True)

    # Return the filtered data to update the DataTable

    return filtered\_data.to\_dict('records')

**Supporting function to prepare Age Data:**

# Convert the 'age\_upon\_outcome' column to a numeric age in weeks

def convert\_age\_to\_weeks(age\_str):

    if pd.isna(age\_str):

        return None

    parts = age\_str.split()

    if len(parts) != 2:

        return None

    number = int(parts[0])

    unit = parts[1].lower()

    if 'year' in unit:

        return number \* 52

    elif 'month' in unit:

        return number \* 4

    elif 'week' in unit:

        return number

    elif 'day' in unit:

        return number / 7

    else:

        return None

df['age\_in\_weeks'] = df['age\_upon\_outcome'].apply(convert\_age\_to\_weeks)

**Callback for Age Distribution Chart:**

# Callback to update the pie chart based on the age distribution

@app.callback(

    Output('graph-id', 'children'),

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

)

def update\_age\_distribution(viewData):

    if viewData is None or len(viewData) == 0:

        return html.Div()

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

    dff['age\_in\_weeks'] = dff['age\_upon\_outcome'].apply(convert\_age\_to\_weeks)

    # Categorize ages

    dff['age\_category'] = dff['age\_in\_weeks'].apply(

        lambda x: 'Younger than 6 Months' if x < 26 else

                  ('Older than 2 years' if x > 104 else f'{x} weeks')

    )

    # Count the occurrences of each age category

    counts = dff['age\_category'].value\_counts()

    fig = px.pie(counts, values=counts, names=counts.index, title='Age Distribution of Dogs')

    return dcc.Graph(figure=fig)

**Callback for Cell highlighting when selected:**

# Callback to highlight a cell on the data table when the user selects it

@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]

**Callback for Geolocation Map:**

# Callback to update the geo-location chart for the selected data entry

@app.callback(

    Output('map-id', "children"),

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

     Input('datatable-id', "derived\_virtual\_selected\_rows")]

)

def update\_map(viewData, index):

    if viewData is None or len(viewData) == 0:

        return html.Div()

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

    # Because we only allow single row selection, the list can be converted to a row index here

    if index is None or len(index) == 0:

        row = 0

    else:

        row = index[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[row, 13], dff.iloc[row, 14]],

                children=[

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

                dl.Popup([

                    html.H1("Animal Name"),

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

                ])

            ])

        ])

    ]

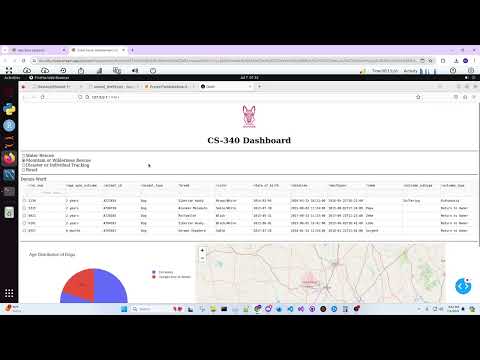
**Run Server:**

if \_\_name\_\_ == '\_\_main\_\_':

    app.run\_server(debug=True)

**Video to Run and review of dashboard:**

[YouTube](https://youtu.be/665FP1tG4qY) : <https://youtu.be/665FP1tG4qY> (requires internet Connection)

[](https://www.youtube.com/embed/665FP1tG4qY?feature=oembed)

Embed: 

**Resources**

1. **MongoDB Installation Guide**
   * [Official MongoDB Installation Guide](https://docs.mongodb.com/manual/installation/)
   * Mastering MongoDB : Alex Giamas. (2022). *Mastering MongoDB 6.x : Expert Techniques to Run High-volume and Fault-tolerant Database Solutions Using MongoDB 6.x: Vol. Third edition*. Packt Publishing.
   * This guide provides step-by-step instructions for installing MongoDB on various operating systems. Citation for Masting MongoDB 6.x Textbook
2. **Python Download and Installation**
   * [Python Official Website](https://www.python.org/)
   * Download and install Python from the official website.
3. **Jupyter Installation Guide**
   * Jupyter Official Installation Guide
   * This guide provides instructions for installing Jupyter Notebook to run and share documents that contain live code, equations, visualizations, and narrative text.
4. **Pip Installation Commands for Required Packages**

Install necessary Python packages using the following commands:

pip install pymongo

pip install dash

pip install jupyter-dash

pip install dash-leaflet

pip install plotly

pip install pandas

pip install matplotlib

1. **Dash Documentation**
   * [Dash Core](https://dash.plotly.com/dash-core-components)
   * [Dash DataTable](https://dash.plotly.com/datatable)
   * Comprehensive documentation for Dash, including guides and tutorials on building interactive web applications with Dash.
2. **Dash Leaflet Documentation**
   * [Dash Leaflet Documentation](https://dash-leaflet-docs.onrender.com/)
   * Documentation for using Dash Leaflet, which allows creating interactive maps in Dash applications.
3. **Plotly Documentation**
   * [Plotly Documentation](https://plotly.com/python/pie-charts/)
   * Detailed documentation for creating interactive plots and charts using Plotly.

**Upcoming Features**

Future enhancements may include:

* Remove hardcoded database connection as this should be more Modular.
* Enhanced security features.

## Contact

Your name: Dennis Ward II