# AAC CRUD README

## About the Project/Project Title

The purpose of this project is to provide an interface for CRUD functionality (Create, Read, Update, Delete) in Python using the MongoDB, specifically with the Austin Animal Center (AAC) database. It provides simple and easy query communication with the database in a user-friendly way. It simplifies the command process for the user.

## Motivation

This search-and-rescue dog dashboard was developed for Grazioso Salvare. The dashboard aims to assist Grazioso Salvare in identifying and categorizing dogs from the Austin Animal Center for search-and-rescue training. It provides interactive filtering options, a data table, and charts to visualize relevant data.

## Getting Started

To get a local copy up and running, a user account with read and write privileges for the AAC database needs to be created. Next, a Python script needs to be created within Jupyter, as well as an IPYNB file to run the test script to verify proper implementation. The Python script needs to create an instance of the class using pre-defined variables for the AAC user and database, then perform the simple CRUD commands with the proper syntax. These commands can then be easily called using simpler queries or entries.

## Installation

This software requires the Mongo Database, Python, and Jupyter. MongoDB is where we store our data and perform our operations, we use Python to bridge the gap between users and commands, and Jupyter can be used for developing Python scripts and testing.

* MongoDB (a document-oriented NoSQL database used as the model component) was chosen as the model component due to its flexibility and scalability in handling document-oriented data. It provides seamless interfacing with Python through the PyMongo library, allowing for efficient data retrieval and manipulation.
* Dash Framework ( a Python framework for building web applications) was utilized to provide the view and controller structure for the web application. It offers a Pythonic way of building interactive web-based dashboards, enabling easy integration with Python data analysis libraries such as Plotly.
* PyMongo (a Python library for interfacing with MongoDB) allows these components to communicate with each other seamlessly.

## Usage

### Code Example – CRUD module

# Create method - C in CRUD

def create(self, data):

if data is not None:

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

return True

else:

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

return False

# Read method - R in CRUD

def read(self, key):

if key is not None:

results = list(self.database.collection.find(key)) # key should be dictionary

return results

else:

raise Exception("Nothing to query, because key parameter is empty")

return []

# Update method - U in CRUD

def update(self, key, data):

if data is not None and key is not None:

numUpsert = self.database.collection.update\_many(key, {"$set": data}).modified\_count # key, data should be dictionary

return numUpsert

else:

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

return 0

# Delete method - D in CRUD

def delete(self, key):

if key is not None:

numDel = self.database.collection.delete\_many(key).deleted\_count

return numDel

else:

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

return 0

**Code Example – Special Query Filters**

# This callback will allow radio buttons to show special query results

@app.callback(

Output('datatable-id', 'data'),

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

)

def on\_selection(value, \*\*kwargs):

if value == 1:

df = pd.DataFrame.from\_records(shelter.read(shelter, {"$or": [

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

{"breed" : "Chesapeake Bay Retriever"},

{"breed" : "Newfoundland"}

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

"$and": [{"age\_upon\_outcome\_in\_weeks": {"$lte": 156}},

{"age\_upon\_outcome\_in\_weeks": {"$gte": 26}}]

}))

elif value == 2:

df = pd.DataFrame.from\_records(shelter.read(shelter, {"$or": [

{"breed" : "German Shepherd"},

{"breed" : "Alaskan Malamute"},

{"breed" : "Old English Sheepdog"},

{"breed" : "Siberian Husky"},

{"breed" : "Rottweiler"}

], "sex\_upon\_outcome" : "Intact Male",

"$and": [{"age\_upon\_outcome\_in\_weeks": {"$lte": 156}},

{"age\_upon\_outcome\_in\_weeks": {"$gte": 26}}]

}))

elif value == 3:

df = pd.DataFrame.from\_records(shelter.read(shelter, {"$or": [

{"breed" : "Doberman Pinscher"},

{"breed" : "German Shepherd"},

{"breed" : "Golden Retriever"},

{"breed" : "Bloodhound"},

{"breed" : "Rottweiler"}

], "sex\_upon\_outcome" : "Intact Male",

"$and": [{"age\_upon\_outcome\_in\_weeks": {"$lte": 300}},

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

}))

else:

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

# Cleanup Mongo \_id field

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

return df.to\_dict('records')

**Required Functionality**

The project includes the following functionality:

* Interactive filter options to filter the Austin Animal Center Outcomes data set by rescue types.
* An interactive data table displaying dynamically filtered data.
* Geolocation chart and a secondary chart dynamically responding to the filtering options.

### Tests

Tests can be run within Jupyter, where an IPYNB file can be created. This file, as shown below, contains the instantiation of an AnimalShelter object, creates some dictionaries for testing purposes, and then calls the Python script’s functions using the testing data to perform specific tasks.

It first adds two new animals to the database, reads them, updates them, reads them again to verify the changes, then deletes them. This can be accomplished because both animals have the same name field, and it verifies that multiple entries can be manipulated at the same time.

Another IPYNB file can be used to launch the dashboard on a local machine, and provides a link to the port in which the application is running. With this, we can view and test the dashboard’s required functionality.

### Screenshots

* Database Import

A screenshot of a computer

Description automatically generated

* New User Creation

A screenshot of a computer

Description automatically generated

* New User Login

A screenshot of a computer screen

Description automatically generated

* Testing Script

A screenshot of a computer

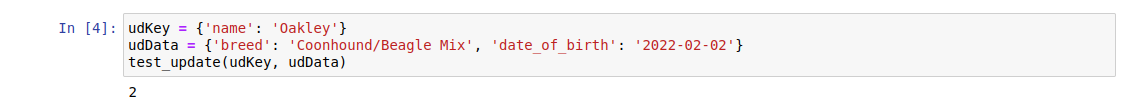
Description automatically generated

*A white rectangular object with red text

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

**

*A screenshot of a computer program

Description automatically generated*

* Starting State of the Dashboard

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

* Water Rescue Filter

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

* Mountain Rescue Filter

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

* Disaster Rescue Filter

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

* Reset Filter (default)

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

## Project Steps

The following steps were taken to complete the project:

1. **Review Dashboard Specifications**: Reviewed the provided Dashboard Specifications Document to understand the project requirements and examples of different dashboard widgets.
2. **Dashboard Development**: Developed the dashboard code using Jupyter notebook (IPYNB file), implementing interactive filter options, data table, and charts.
3. **Database Setup**: Set up MongoDB as the database to store and retrieve data for the dashboard.
4. **CRUD Python Module**: Developed CRUD (Create, Read, Update, Delete) Python module from Project One to interact with the MongoDB database and retrieve data.
5. **Interactive Filter Options**: Created interactive filter options (radio items, drop-down menu) to allow users to select data based on filtering functions.
6. **Dashboard Widgets**: Modified or created dashboard widgets (data table, charts) to respond dynamically to updates from the interactive filter options.
7. **Testing and Deployment**: Tested the functionality of all components of the dashboard and deployed it by running the IPYNB file.

**Challenges and Solutions**

Challenges Encountered:

* Ensuring seamless integration between MongoDB and Python code.
* Implementing dynamic updates in the dashboard widgets based on filter options.

Solutions:

* Thorough testing and debugging of the CRUD Python module to ensure proper interaction with MongoDB.
* Utilizing the Dash framework's callback functionality to update the dashboard widgets dynamically based on user input.

## Roadmap/Features

The following features would continue to build upon this project, and will be continually implemented:

* An update to the pie chart, which allows the raw data with many different kinds of breeds in the system to be viewed, will help this visual data’s value by cleaning up the categories.
* A button that provides the user access to the database aggregation function. The function is written and has been ran to clean up the database (delete duplicates), but it is currently unable to be run within the dashboard.

## Contact

Morgan Getkin