# AnimalShelter README

## About the Project/Project Title

AnimalShelter performs the CRUD (Create, Read, Update, Delete) functions on the Austin Animal Center database. This is written in Python and works directly with the MongoDB by using the PyMongo module.

## Motivation

This project is being written to help the “aacuser” database user perform the CRUD functions in a more controlled and efficient manner. This program helps to make the administration of the Austin Animal Center database easier.

## Getting Started

In order to get started using this program, you’ll need to create a MongoDB database and a collection that matches what is in the program (currently the database is called AAC with a collection called animals). You’ll also need to create a user with readWrite access to the database (the user is currently called “aacuser” with a password of “aacuser”). Ensure that the database that the user is set up to authenticate against is correct in the \_\_init\_\_ method (the aacuser user authenticates against the AAC database).

The create() and read() methods were created to expect a dict to be passed as an argument. These use the insert() (for create) and find() (for read) pymongo functions to perform the actions. The collection name will need to be changed in these methods if your configuration doesn’t match (my collection is called animals).

## Installation

To use this program, you’ll need to install the following:

MongoDB – instructions for your OS can be found at the link below

[Install MongoDB — MongoDB Manual](https://docs.mongodb.com/manual/installation/)

Python 3.x – this program was written using version 3.6.9

[Download Python | Python.org](https://www.python.org/downloads/)

Jupyter Notebook – Installation instructions can be found at

[Installation — JupyterLab 3.0.16 documentation](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)

## Tools

The tools that I used to create this program were:

MongoDB – this database option was used as it is easy to access using the Pymongo API and its methods.

Jupyter – this was used for testing of the Python program and allowed me to test each method to ensure that it works as designed.

Pymongo – This Python module includes tools for working with MongoDB. This is the recommended way to work with MongoDB from Python.

Dash – Dash was used as it provides templates and features that allows for easy development of components and UI elements using Python. It also interacts with other open source languages to create feature-rich application dashboards.

## Usage

### Connect to DB

#### Code Example

*def \_\_init\_\_(self, user:str, pw:str):*

*# Initializing the MongoClient. This helps to*

*# access the MongoDB databases and collections.*

*self.client = MongoClient('mongodb://%s:%s@localhost:43345/AAC' % (user, pw))*

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

#### Test

from animalShelter import AnimalShelter

shelter = AnimalShelter("aacuser","aacuser")

#### Screenshots

****

### Create document in the database

#### Code Example

*def create(self, data: dict) -> bool:*

*"""Insert a document based on a provided dict"""*

*if data is not None:*

*result = self.database.animals.insert(data) # data should be dictionary*

*print(result)*

*if result is not None:*

*return True*

*else:*

*return False*

*else:*

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

#### Test

*shelter.create({"age\_upon\_outcome":"1 year","animal\_id":"Z000000","animal\_type":"Dog","breed":"Golden Retriver","color":"Red","date\_of\_birth":"2021-03-12","name":"Rocky","outcome\_type":"Transfer"}).*

#### Screenshots

*Text

Description automatically generated*

### Read

#### Code Example

*def read(self, data:dict):*

*"""Find a document based on a provided dict"""*

*if data is not None:*

*return self.database.animals.find(data) # data should be a dictionary*

*else:*

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

#### Tests

*shelter.read({"animal\_id":"Z000000"})*

#### Screenshots

**

## Dashboard Features

To use the dashboard, install Jupyter Notebook using the link in the Installation section. The created dashboard presents the necessary data in an easy-to-use implementation and gives access to requested functionality for quick searches.

**Start page**

The start page shows an unfiltered view of the dashboard data and the available options for filtering. Each page shows the Grazioso Salvare logo along with my unique identifier. The data table is shown as well as a pie chart with the breakdown of the different breeds in the table and a map with a pinpoint of the location one of the animals. There is a dropdown list just above the table which includes options for filter by Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking and a Reset to return to an unfiltered view.

A picture containing table

Description automatically generated

Graphical user interface, application

Description automatically generated

**Water Rescue**

The Water Rescue option displays the following attributes:

* Breeds – Labrador Retriever, Chesapeake Bay Retriever, Newfoundland
* Sex – Intact Female
* Age – 26 to 156 weeks

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Mountain or Wilderness Rescue**

This filtering option provides the dogs with the following attributes:

* Breed – German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, Rottweiler
* Sex – Intact Male
* Age – 26 to 156 weeks

Text

Description automatically generatedGraphical user interface

Description automatically generated

**Disaster or Individual Tracking**

This filter displays animals with the following attributes:

* Breed – Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler
* Sex – Intact Male
* Age – 20 to 300 weeks

A picture containing text

Description automatically generatedGraphical user interface, application

Description automatically generated

**Reset**

This filter resets the display to the unfiltered data.

A picture containing text

Description automatically generatedGraphical user interface, application

Description automatically generated

## Steps to complete the project

This project was completed by first accessing the data that was provided in a .csv file and importing it into a MongoDB collection. Once that data was available, I tested the usage of the MongoDB using the Mongo shell and viewed the types of data that was included in the collection. I wrote the Python module using the available PyMongo API to create the CRUD functions into an importable module which was then used in the dashboard. The dashboard was by following the Requirements Document and adding the features and elements that were requested.

## Challenges in the Project

The largest challenge that I faced was in implementing the map as I wasn’t correctly providing all of the data from the documents in my Read() function due to an earlier change. After researching the Dash datatable components, I realized my error and reverted the change to allow the Read() method to provide all of the data for the requested documents.

Another challenge that I overcame was that many of the animals have multiple breeds or the breed name isn’t the same in all places. To overcome this challenge, I imported the ‘re’ Python module to allow me to use regular expressions in the queries. This enabled me to catch all types and versions of the specified breeds for the dropdown menu filtering.

I initially also had a problem with the pie chart not updating with the filters from the dropdown menu but realized that if I used the same type of data as I did in the map, that would give me the updated data that I needed. A related issue is that the pie chart for the unfiltered view has many breeds with very low percentages that would look better if they were grouped into a section for ‘Others’ but I was unable to get that functionality to work.

## Contact

Your name: Greg Roberge