# CS 340 README

## About the Project/Project Title

This project is a python module that interacts with a MongoDB database. Specifically, the module can log in to the database and effectively perform CRUD operations onto a collection within the database. There is also a dashboard that can graphically interact with the database.

## Motivation

The project was born from a company, Grazioso Salvare, requesting an application that can categorize and identify dogs for specific search-and-rescue training programs. The database comes from a CSV file containing data from a combination of animal shelters and loaded into a MongoDB collection.

## Getting Started

To run this project locally, first ensure you have admin access to a MongoDB installation (see [Installation](#_Installation)). Afterwards, download the [Austin Animal Center Outcomes CSV](https://doi.org/10.26000/025.000001) file to an accessible location on your machine.

Once the CSV is downloaded, import the data into your MongoDB, with the database name “aac” and collection name “animals” using the following command. Note: Be sure to insert the name and location of the CSV file you downloaded on your machine:

*mongoimport --db* ***aac*** *--collection* ***animals*** *--drop --file* ***location/to/csv/file.csv*** *--type csv --headerline*

If done correctly, your terminal should display something like the following:

A computer screen shot of a computer screen

Description automatically generated

Now, open an admin MongoDB shell and create a new user account with read and write permissions to the new database using the following commands. Note: Be sure to remember the username and password you use in this step for later:

*mongosh*

*use admin*

*db.createUser({*

*user: “USER ACOUNT NAME”,*

*pwd: passwordPrompt(),*

*roles: [{*

*role: “readWrite”,*

*db: “aac“*

*}]*

*})*

Once the new account is created, download the CRUD\_Python\_Module.py file to your machine. Within the initializer method, change the variables **USER** and **PASS** to the username and password you created the new account with from before.

In the module, the create method takes in a dictionary that consists of data for one item to be inserted into the “animals” collection. The method returns a Boolean if that insertion was successful. The primary challenge of developing this method was verifying if the item was inserted into the collection. By researching common techniques on how to perform such a task in MongoDB, this obstacle was easy to bypass.

Likewise, the read method within the module takes in a dictionary that is formatted similarly to a native MongoDB shell query when using the *find* command. The dictionary is passed to the *find* API from the pymongo library, and the method returns a list of results. The main challenge when developing this method was converting the results cursor into a Python list efficiently. By reading through the pymongo documentation, the method of converting a cursor into a Python list was discovered.

Similarly to the previous method, the update method accepts a dictionary that is formatted as a *find* query, as well as another dictionary formatted as a MongoDB shell document for the *updateMany* command. These dictionaries are passed to the *update\_many* API from the pymongo library, and the method returns the number of documents that were updated given the query and update operation dictionaries. The main challenge of developing this method was deciding which update API to use. After carefully studying how the client, Grazioso Salvare, may use this functionality, it was determined that the client will most likely want to update multiple documents within the database as efficiently as possible.

Finally, the delete method accepts a dictionary that is formatted as a MongoDB *find* query, which is passed to the *delete\_many* API from the pymongo library. The number of documents removed from the database is returned as a result of executing this method. Similar to the update method, deciding whether to use the *delete\_one* or the *delete\_many* API was the hardest challenge when developing this method. Again, after thoroughly reviewing the client’s potential use cases of this functionality, it was determined that removing multiple documents within one method call was the most likely use case when using this functionality.

## Installation

To get straight into the project, ensure you have access to the following tools and Python libraries:

* MongoDB 7.0.21 server
* mongoimport version 100.13.0
* Python 3.11.2
* JupyterLab
* PyMongo library
* Dash library
* Dash-leaflet library
* Plotly library
* Pandas library

Because the structure of the animal shelter data is not entirely the same from record to record, MongoDB 7.0.21 is used within this project. Read [Install MongoDB Community Edition](https://www.mongodb.com/docs/manual/administration/install-community/?operating-system=linux&linux-distribution=red-hat&linux-package=default&search-linux=with-search-linux) to install a local MongoDB server on your machine.

The mongoimport version 100.13.0 is used to import the animal shelter CSV data into the database. Depending on how the MongoDB server was installed, you may need to [install the MongoDB Database Tools](https://www.mongodb.com/docs/database-tools/installation/?operating-system=linux&package-type=deb) separately.

Due to its ease of use and data-processing capabilities, Python 3.11.2 is used. The documentation for installation can be found [here](https://www.python.org/downloads/release/python-3112/).

To test and interactively use the project module, JupyterLab is used. This tool can be installed from [here](https://jupyter.org/install).

To interact with the MongoDB server using Python, the PyMongo library is used. It can be imported using the following command:

*pip install pymongo*

The dashboard is a Dash application, which is a framework for building data visualization apps within Python. The Plotly library is used to display the pie chart of the percentage of animal breeds per selected filter. To aid in displaying the geolocation of a selected entry, the Dash-leaflet library is used. These libraries can be installed using the following commands:

*pip install dash*

*pip install plotly*

*pip install dash-leaflet*

The Plotly library requires the installation of the all-powerful data analysis and manipulation tool, Pandas. The library can be installed using the following command:

*pip install pandas*

## Usage

### Code Example

To create a new handler that connects to the database, import the AnimalShelter class from the module and initialize a new instance of AnimalShelter by passing in the username and password of the user account created (see [Getting Started](#_Getting_Started)):

*from CRUD\_Python\_Module import AnimalShelter*

*username = 'aacuser'*

*password = 'helloWorld'*

*shelter = AnimalShelter(username, password)*

To insert a new record into the database, create a dictionary containing a record’s data and use the AnimalShelter instance’s *create* method. The method returns **true** if the insertion was successful, false otherwise:

*new\_record = {‘animal\_id’ : ‘99999’, ‘animal\_type’ : ‘cat’}*

*success = shelter.create(new\_record)*

To find records with specific data, create a dictionary containing MongoDB query parameters and pass it into the AnimalShelter instance’s *read* method. A list of results will be returned, which will be empty if no records satisfy the query:

*query = {‘animal\_type’ : ‘cat’}*

*results = shelter.read(query)*

To update records, create a dictionary containing MongoDB query parameters and a dictionary containing MongoDB update operation parameters. Pass these dictionaries into the AnimalShelter instance’s *update* method. The number of updated records will be returned:

*query = {'animal\_type': ‘cat’}*

*operation = {'$set' :*

*{'outcome\_type' : 'relocate'}*

*}*

*updated\_count = shelter.update(query, operation)*

To delete records, create a dictionary containing MongoDB query parameters and pass this dictionary into the AnimalShelter instance’s *delete* method. The number of deleted records will be returned:

*query = {'animal\_type' : ‘cat’}*

*deleted\_count = shelter.delete(query)*

**Dashboard**

The dashboard, as its most basic functionality, displays entries of animals and their respective data obtained through the database. The dashboard displays the percentage of each dog breed given a selected filter in the form of a pie chart. Since each animal contains information about their location, the dashboard also displays a map with a marker indicating a selected animal’s recorded location in the world. See the section [Screenshots](#_Screenshots) to view the different functionality of the dashboard.

The default view of the dashboard lists all entries within the database with one entry preselected and location displayed on the map. The pie chart displays the percentage of the top 10 animal breeds in the database, with the remaining breeds labeled as *other*.

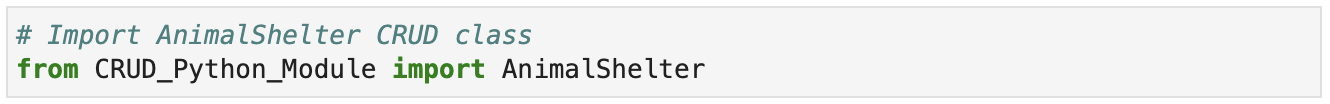
A user can select a filter, which is a specific type of search-and-rescue, located at the top of the data table. Each filter retrieves only the preferred dogs given their sex, breed, and age range deemed appropriate by the client for the specific type of search-and-rescue labeled by the filter.

### Tests

You can test the module by loading the .ipynb file into JupyterLab and running each cell in sequence.

### Screenshots

Import the module



Create a new instance of the AnimalShelter class

A close-up of a computer screen

Description automatically generated

Insert a new record into the database

A screenshot of a computer code

Description automatically generated

Find records in the database that match a specific query

A screenshot of a computer

Description automatically generated

Update records in the database that match a specific query

A screenshot of a computer program

Description automatically generated

Delete records in the database that match a specific query

A screenshot of a computer

Description automatically generated

Starting and reset state of the dashboard

A screenshot of a computer

Description automatically generated

Filter for preferred water rescue breeds in the dashboard

A screenshot of a computer

Description automatically generated

Filter for preferred mountain or wilderness rescue breeds in the dashboard

A screenshot of a computer

Description automatically generated

Filter for preferred disaster rescue or individual tracking breeds in the dashboard

A screenshot of a computer

Description automatically generated

## Roadmap/Features

In the future, it is planned to utilize a better pattern-matching technique, such as regex, to search for appropriate dog breeds within the database.

## Contact

Your name: Justin Crouch