# Byron Laferriere

CS-340

23 JUNE 2021

## About the Project/Project Title

This project is aimed to provide both a database service and front-end user interface. The purpose of this project is to provide a quick option to query for animals that are up for adoption, to be selected for search and rescue training. This will cover the creation of a non-relational database, MongoDB, that can be managed through a Python interface named Pymongo. Creating CRUD operations for employees at the shelter to use, will allow for the database to be accessed and updated as needed. The goal is to provide an exceptionally quick Mongo database that can be edited easily through Python, using Pymongo, for Grazioso Salvare employees to manage and access necessary information in choosing the correct candidates for the search and rescue teams.

## Motivation

This project is motivated by the possibility of increasing the available information for pets that have passed through the animal shelter. Grazioso Salvare is seeking to increase their functionality and efficiency in searching for candidates, by combining MongoDB and Python to create a database that can have CRUD operations performed on it. This will simplify the client server search and ease the search process.

## Getting Started

To get a local copy up and running, some prior knowledge of how to operate databases will be necessary. A .csv (comma separated value) file will need to be downloaded, which contains the database. Once a non-relational database, like MongoDB has been installed, the .csv file can be imported through commands. Next, user authentication can be set up, which is necessary to link Python with. This authentication is used in coding and the project is nearly complete.

## Installation

For a successful completion, the following software and tools must be installed:

* MongoDB- Community Edition or a higher tier must be used.
* Pymongo Driver- Used for communication between Mongo and Python.
* Python 3+- Use the IDLE editor to create scripts to be run.

## Usage

This new library will provide employees with a massive database, that is easy to operate and access. Below is an example of how easy it was to load 10,000 files into the database to be managed. Using quick commands like this, showcase how beneficial a project like this can be.

*Text

Description automatically generated*

Now that the database has been created, the administrator can easily begin creating user authentications for employees. The code example below showcases how easy it is to create new users and provide access to specific databases, or all databases, as well as what capabilities these users have with the files.

A screenshot of a computer

Description automatically generated

With this new ‘aacuser’ username and password created, we can now access the database through python scripting. Below, code is attached that exemplifies how to begin connecting the Python interpreter to MongoDB, using the pymongo driver and assigned username and password.

Graphical user interface, text, application, email

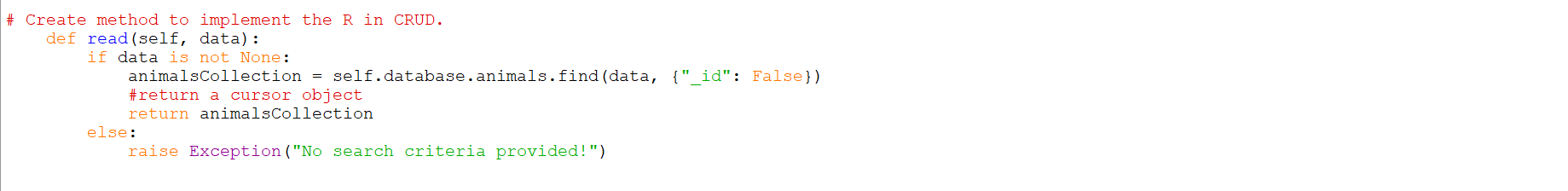
Description automatically generated

Now, the CRUD operations can be implemented in our AnimalShelter class. First, we will implement **Create** command. This command takes a dictionary as its argument, then returns with a Boolean value if false or successful.

Text

Description automatically generated with medium confidence

Moving on to the **Read** method in the CRUD operations, we will implement this method in a similar manner. Using the read command requires taking a dictionary as an argument and returning a cursory object that contains the query results. Below, I created implement this method and return the results of my search.



Next, comes the **Update** method in the CRUD operations. We will use the update command, which also requires a dictionary to be an argument and then return the count of all modified documents that met our criteria we input. The code below shows an example of the creation of an update method for our AnimalShelter class.

Text

Description automatically generated with medium confidence

Lastly, we have the **Delete** method to define in our CRUD operations. The delete command requires a dictionary as an argument and then returns a count of the items deleted by the command. Below is an example of how to create the delete method.

Graphical user interface, text, application

Description automatically generated

### Tests

With our code now written, which can access our database, we can now begin testing whether or not our code works. To do so, we will need to test our operations that we have added to the Animal Shelter class. The first test that we will run is intended to test the input of a new animal into the database. Once the animal is **created**, a cursor points to the new location of that animal in the database. Using the ‘from command’ we are able to import our code that contains the access to the database.

Text

Description automatically generated

Next, we need to test the program for a **read** capability. The next snippet of code exemplifies through a test accessing the Animal Shelter class, just how easy it is to locate an animal or group of animals.

A picture containing text

Description automatically generated

Now we need to test our **update** command to ensure that it is working also. This next snippet of code displays how I tested the animal\_shelter.py script to see if it could successfully access the database and update all matches for the query.

Text

Description automatically generated

Lastly, we need to test the **delete** command to verify that it is working as well. To do this, the following test was run through the MongoDB to find a match for the query and delete them. After this has been accomplished, the system is to tell you how many files were deleted.

**CREATING THE DASHBOARD**

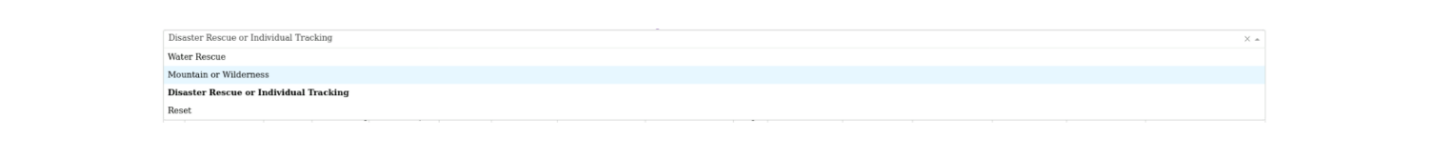
The next section of this README will focus on the creation of the Dashboard for the application. The Dash framework within Jupyter was selected for this project. Dash is an open-source Python library used to create web applications. Dash easily creates GUI ‘s for clients using its own built-in components and html objects. One of the best features of Dash, is the ability to write in callback’s to constantly update the dashboard.

I have attached a screenshot of the display for the dashboard for preview.

A picture containing text

Description automatically generated

Below is an example of the interactive dropdown menu used in the program.



Next is an example of the sorting capabilities of the dashboard, in combination with the database.

Graphical user interface

Description automatically generated

The next example shows the Leaflet geo-map that was intertwined in this project.

Map

Description automatically generated

And lastly for the dashboard we also integrated a pie chart for visualizations. Charts like this are available through the dash core components package.

Chart, pie chart

Description automatically generated

**PROOF OF CONCEPTS/EXECUTION**

To showcase just how this dashboard works, with all of its loaded features, I have gone through categorically and selected each type of rescue. This will highlight the capabilities of the display and show why it will be useful.

*No Selection Made:*

A picture containing text

Description automatically generated

*Dropdown changes to water rescue:*

When the dropdown menu has the water rescue option selected, the dashboard will update via callbacks and produce the screen below.

A picture containing chart

Description automatically generated

## *When Mountain/Wilderness Rescue selected:*

When the dropdown menu has the mountain/wilderness rescue option selected, the dashboard will update via callbacks and produce the screen below.

A picture containing chart

Description automatically generated

*When dropdown is changed to DR/Individual Recovery:*

When the dropdown menu has the disaster rescue/individual recovery option selected, the dashboard will update via callbacks and produce the screen below.

A picture containing chart

Description automatically generated

*When Reset is selected in the dropdown menu:*

When the dropdown menu has the reset option selected, the dashboard will update via callbacks and produce the screen below.

A picture containing application

Description automatically generated

**How to create the dashboard:**

When creating this dashboard, the first thing to do was to establish the connection to the database which contained the data on the animals needed. This involved creating user authentication for different users accessing the program. Next, the querying of the database was required to begin the population of the dashboard. To do this, you take the queried information and create a data table using the Pandas module. Once this is done, you can create a dictionary out of the data table, which is to be read and accessed as needed by the dashboard. The next step I followed was creating html layout objects for the header, client picture, and link to their website. After this, reactive objects must be set up, this includes the callback routines that update the objects upon changes. The reactive objects included, my dropdown selection menu, a datatable for selection, and the geo-map and pie chart for visualizations. The results of the query are evaluated when selected through the dropdown menu, which then determined the output to be displayed.

**Challenges:**

The biggest challenge I faced was the new IPYNB files being used every week. Instead of continuous documents that had the bugs fixed from week’s before. We were handed new files to work with, when they only added a new line or two of code, that could have easily been supplemented into existing code. This set me back on trying to blend old code that was written, with the new pieces being added weekly. If anything, it taught me the importance of checking code thoroughly for errors before combining with different/working pieces. Other than that, I was challenged by the complexity of the different options that could be used within Dash for the creation choices.

## Contact

Byron Laferriere