# CS 340 Project Two README

By Howard GIlmore

## The Project Subject

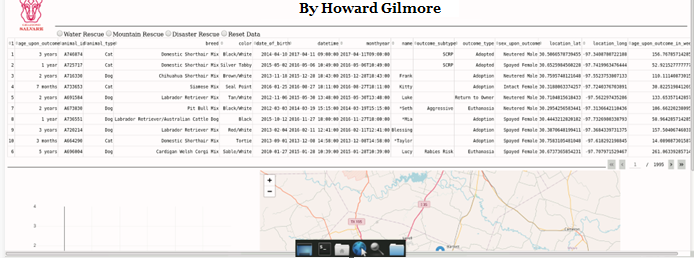
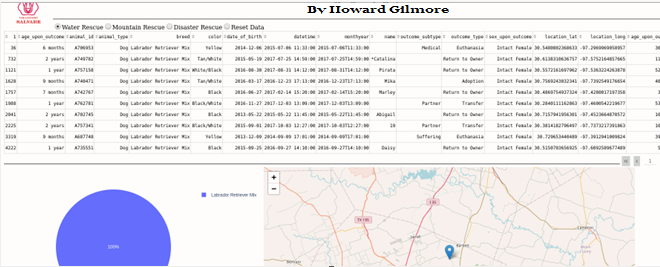
## Grazioso Salvare, an international rescue and training organization for animals, is in need of a system to categorize canines based on information gathered from shelters. Grazioso Salvare contracted Global Rain to build the app's backend database and user interface. With the dashboard, users may interact with the database like a table. The first animal on the list may be found with the help of two widgets, and the influence of data filters on animal distribution can be shown with the help of another.

## Required Functionality

1. The software in issue will credit both the dashboard's designer, Howard Gilmore, and its developer, Grazioso Salvare (whose logo will link to their website at www.snhu.edu).:

Graphical user interface, application

Description automatically generated

1. The filter choices shown as radio buttons up top are interactive and have an effect on the behavior of this dynamic program. The data table's contents and the widgets' functionality are both controlled by these filters. There's a map indicating where the first animal is located that's presented in the data table, and there's a pie chart showing how many different breeds are shown for each filter option. Below is a screenshot of the dashboard in its unfiltered, initial condition.
2. The client's chosen breeds are used to restrict data searches for Water Rescue dogs. When a filter is applied, the remainder of the dashboard is refreshed to display the results:
3. Inquiries for Mountain or Wilderness Rescue dogs are put through a filter that takes into account the client's preferred breed of dog. After you apply a filter, the remainder of the dashboard will automatically update to show you the results, which are as follows:Graphical user interface, application

   Description automatically generated
4. Inquiries about Disaster or Individual Tracking Rescue dogs are sent via a data filter that takes into account the breeds that the client has indicated they are interested in. The remaining components of

the dashboard have been changed to reflect the data that have been filtered, and they are as follows:Graphical user interface, application

Description automatically generated

6. Finally, the radio button option "Reset Data" does as it is indicated and resets the data table as well as the widgets to their initial, unfiltered state:Graphical user interface, application

Description automatically generated

## Tools and Installation

* MongoDB: <https://docs.mongodb.com/manual/installation/#mongodb-community-edition-installation-tutorials>
* o If you follow the lessons that have been provided above, you will get an understanding of how to install MongoDB on Windows, Linux, and MacOS.
* o MongoDB is used to store the data from many different animal shelters and also provides CRUD capabilities on the data that is stored. Pymongo was used to access MongoDB (see below for more information), which made it possible to use the tools for Python programming. The bulk of this application's backend development was accomplished via the use of these technologies.
* Dash Framework and Components: <https://dash.plotly.com/>
  + Dash is a Python framework for developing online analytic apps, as stated on the official website. Dash is ideal for building data visualization apps with highly custom user interfaces in pure Python. Those who use Python for data work will find it very useful. Using this framework, designers may build applications with dynamic user interfaces Making use of the Nucleus of Dash. The data from the animal shelter may then be shown in a variety of interactive formats including tables, charts, and graphs that respond in real time to the user's selections.
  + To install Spyder, go to: https://docs.spyder-ide.org/current/installation.html.
  + Pymongo is a set of Python drivers for the MongoDB database; the installation instructions for it can be found at https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/. o The above link shows how to install Spyder, the Python 3.6 IDE, on Windows, Linux, and MacOS. https://pymongo.readthedocs.io/en/stable/installation.html
  + The following links provide examples of the import lines required to use all of MongoDB's features in Pymongo. This opens the door for Python users to access and modify MongoDB databases. Remember to include "from pymongo import MongoClient" at the very beginning of your CRUD classes and test scripts before you dive into any backend programming.
  + You may learn more about ObjectID here: https://pymongo.readthedocs.io/en/stable/api/bson/objectid.html.
  + o Since our CRUD class is an object, make sure that your CRUD class and testing script for backend development include the import command: "import bson.objectid import ObjectId."
  + - Jupyter Notebook: https://jupyter.org/install o For installation instructions, please refer to "Getting started with the traditional Jupyter Notebook." In addition to serving as the integrated development environment (IDE) for front-end development, this tool will be utilized to verify the correct operation of our back-end CRUD class. Import dumps: https://www.geeksforgeeks.org/json-dumps-in-python/ is a tool used to develop and link the backend and frontend components of a web application.
  + To transform Pymongo objects into JSON strings, prefix your CRUD file with "from bson.json util import dumps."
  + Unit Test - https://docs.python.org/3/library/unittest.html
  + o Unit testing is a simple method to assure code functioning; to use its features for backend development, just put "import unittest" at the start of your testing script.

## Steps to Complete

Setting up this project in a local environment involves both backend and frontend development, shown within the following steps:

1. First and foremost, we must develop the backend functionality of this application. To do so, install MongoDB which houses the data itself as well as provides the functions necessary for the project. See **Installation** for links to access MongoDB as well as tips on the installation process.
2. Next, import the necessary data into MongoDB, which is in the form of the file “aac\_shelter\_outcomes.csv”. This can be imported by entering the /usr/local/datasets/ directory within the terminal, then typing “mongoimport --port [YOUR PORT NUMBER] --db AAC --collection animals --type=csv --headerline ./aac\_shelter\_outcomes.csv”
3. After successfully importing the data, the following link illustrates how to enable user authentication for the database: <https://docs.mongodb.com/manual/tutorial/enable-authentication/> Create an admin account by following steps #2-3, then re-start MongoDB and authenticate yourself as the admin with the following command: “mongo --port [YOUR PORT NUMBER] --authenticationDatabase "admin" -u "myUserAdmin" -p”
4. When you're ready to add more users for the client business, follow the instructions in the aforementioned link's steps #6-7 to set up a user account named "aacuser" with a password of your choice.
5. Install a Python integrated development environment; we're using Spyder (Python 3.6) here; for details, check out Installation.
6. Make sure you've imported MongoClient before beginning to write the CRUD class's code for creating and reading data from the database (again, see Installation). Remember that your MongoDB port number, the database you're accessing (AAC), and your login and password all need to be entered in the CRUD class's startup function. Your group won't have access to the necessary information until you make these changes.
7. In a manner analogous to "insert" inside MongoDB itself, the Create method first gets data from the user to insert, then verifies that the data is not empty, and then puts the data into the database if it is not already there.
8. The Read approach is organized like a question, which is the eighth distinguishing feature. In response to a user's search query, the Read method employs a "find" command to locate relevant documents. An error is generated if the criterion is null since it is impossible to read null data.
9. There is a query-like structure to the ReadAll procedure, which brings us to the number nine. The data table utilized in the development process is populated with the information retrieved using this approach. To guarantee that all records are retrieved without any hitches, you should form your query as follows: database.collection.find(data, "\_id": False).
10. The Update method implements the "update" command from MongoDB and takes two arguments. The first one requests the updated document or documents, while the second one stores the modified version of the requested document. An exception is thrown if the arguments are empty or the update fails.
11. The Delete operation makes use of MongoDB's "remove" command. A delete query matching the target document(s) is sent to this method. An exception is thrown if the query returns no results or the delete fails.
12. We must now complete the application's backend by testing it. Tests should be built to ensure the correct operation of each CRUD method in your new CRUD class. To write unit tests for each method in the CRUD class, the developer should use a tool like Jupyter Notebook and import the CRUD class, unittest, and the MongoClient.

13. Following this, we will go on to the frontend of the application in Jupyter Notebook, where you will instantiate the CRUD class you created and import the Dash frameworks and Pymongo that will be used.

14. After the user has been verified, the dashboard's design may begin. Get started on making a header by importing the logo file used by the client and adding a title and identification.

15. Once you have that done, make a list of the Radio Items that will act as filters for the information. Included in this process are the Dash Core Components of Water Rescue, Mountain Rescue, Disaster Rescue, and Reset Data.

16. Now, you should create an active data table that includes information from all three sanctuaries. Do this by using the ReadAll function on your CRUD object.

Create a pie chart and a geolocation chart using the Dash Core Components. To do this, we make use of a set of callbacks that are tied directly to the table of data. When a filter is applied to the data table, the widgets will automatically reflect the new results.

17. Finally, construct queries depending on the client's needs to make sure the radio buttons may be used as filters. A connection to MongoDB is established using the Read method of the CRUD class, and the data table and widgets are filtered depending on the queries provided.

## Challenges and Solutions

When building this app, one of the biggest issues was getting all of the data from the MongoDB dataset into the Dash data table. On sometimes, Dash's framework would misinterpret the results of my "retrieve all" queries. This led to an endless loading screen that never completed. Adding the projection syntax to my CRUD ReadAll function allowed me to avoid sending "\_id" variables and therefore address the issue. My CRUD class's ReadAll function was modified to read "database.collection.find(data, "\_id": False"). A few more issues arise afterwards, Therefore majority of the time I have successful runs but I noticed if I attempted to run it on another platform it would struggle, I’m still working those bugs out but if you have any issues running this on your client please let me know.

## Contact

Howard Gilmore

Howard.gilmore@snhu.edu