# CS 340 Project Two README

By Sarah Deleppo

## About the Project/Project Title

An innovative international rescue-animal training company, Grazioso Salvare, is seeking a software application that can work with existing data from the animal shelters to identify and categorize available dogs. Global Rain has contracted for a full stack development of this application that will include a database and a client-facing web application dashboard, through which users at Grazioso Salvare will access the database as a data table as well as visualize the data through interactive widgets. The interactive widgets include a geolocation chart showing the location of the first animal in the list and a pie chart illustrating the variety of breeds based on the filtered data.

## Required Functionality

1. The application in question will include the **Grazioso Salvare logo** that links to their homepage (www.snhu.edu) as well as **the creator of the dashboard**, Sarah Deleppo:

Text

Description automatically generated with medium confidence

1. Graphical user interface

   Description automatically generatedThis application is dynamic and responds to interactive filter options shown as radio items above. These filters both delegate the information within the data table as well as the interactive widgets. These widgets include a geolocation chart showing the location of the first animal listed in the data table, as well as a pie chart illustrating the variety of breeds displayed per filter. The following image is the **starting state** of the dashboard without any filters applied:
2. Graphical user interface, application

   Description automatically generatedThe data filter for **Water Rescue** dogs’ queries based on the preferred breeds (Labrador Retriever Mix, Chesapeake Bay Retriever, Newfoundland), sex (Intact Female), and training age (26 weeks to 156 weeks) as specified by the client. The rest of the dashboard updates to view this filtered data:
3. Graphical user interface, application

   Description automatically generatedThe data filter for **Mountain or Wilderness Rescue** dogs’ queries based on the preferred breeds (German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, Rottweiler), sex (Intact Male), and training age (26 weeks to 156 weeks) as specified by the client. The rest of the dashboard updates to view this filtered data:
4. Graphical user interface, application

   Description automatically generatedThe data filter for **Disaster or Individual Tracking Rescue** dogs’ queries based on the preferred breeds (Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler), sex (Intact Male), and training age (20 weeks to 300 weeks) as specified by the client. The rest of the dashboard updates to view this filtered data:
5. Graphical user interface, application

   Description automatically generatedLastly, the radio item **Reset Data** does as described, and returns both the data table and widgets to their original, unfiltered state:

## Tools and Installation

* MongoDB: <https://docs.mongodb.com/manual/installation/#mongodb-community-edition-installation-tutorials>
  + Following the above tutorials offers insight on how to install MongoDB on Windows, Linux, and MacOS
  + MongoDB is used to house the data from various animal shelters as well as provide CRUD functionality on said data. MongoDB was accessed using Pymongo (see below) to allow for Python development using the tools. These tools comprised the majority of the backend development for this application.
* Dash Framework and Components: <https://dash.plotly.com/>
  + According to Dash’s site, “Dash is a productive Python framework for building web analytic applications. Dash is ideal for building data visualization apps with highly custom user interfaces in pure Python. It's particularly suited for anyone who works with data in Python.” This framework provides developers access to interactive user interfaces utilizing Dash Core Components. This allows us to display the animal shelter data as a series of tables, charts, and graphs and allow these components to dynamically update based on the filters we provide.
* Spyder: <https://docs.spyder-ide.org/current/installation.html>
  + The above link illustrates installation of Spyder, the Python 3.6 IDE, on Windows, Linux, and MacOS
* Pymongo- Python MongoDB drivers:

<https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/> <https://pymongo.readthedocs.io/en/stable/installation.html>

* + The above links illustrate the import statements needed to have full access to MongoDB functionality using Pymongo. This allows users access and editing capabilities to databases within MongoDB using Python. At the top of your CRUD class and test script for backend development, ensure you have the line: “from pymongo import MongoClient”
* ObjectID: <https://pymongo.readthedocs.io/en/stable/api/bson/objectid.html>
  + Since our CRUD class is an object, ensure you have the import command: “import bson.objectid import ObjectId” in both your CRUD class and testing script for backend development
* Jupyter Notebook: <https://jupyter.org/install>
  + See “Getting started with the classic Jupyter Notebook” for installation instructions. This tool will be used to test the functionality of our CRUD class for backend development as well as provides the IDE for frontend development. This tool is used to write and connect the backend and frontend components into one functioning web application
* Import dumps: <https://www.geeksforgeeks.org/json-dumps-in-python/>
  + Include the line: “from bson.json\_util import dumps” at the top of your CRUD file in order to convert Pymongo objects to JSON strings
* Unit Test: <https://docs.python.org/3/library/unittest.html>
  + Unit testing is an easy way to verify functionality of code, ensure you include “import unittest” at the beginning of your testing script to have access to its functionality for backend development

## 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. Next, create additional users for the client company, follow the above link steps #6-7 to create a user account called “aacuser” with the password of your choosing.
5. Next, install the Python IDE of your choosing, in this instance we are using Spyder (Python 3.6), see **Installation** for access.
6. Begin writing your code for the CRUD class, to Create and Read within the database, ensure that you have imported MongoClient within your code (again, see **Installation**). Note that the authentication to MongoDB is in the *initialization* method for the CRUD class and should include your port number, the database you are accessing (AAC) and your username and password. Without these crucial additions, your class will be unable to access the desired data.
7. The **Create** method follows a similar framework to “insert” within MongoDB itself, the method receives data from the user to insert, then checks this data is not empty, and if not, the method inserts said data into the database.
8. The **Read** method follows a query-like structure. Upon receiving the criteria the user intends to search for, the Read method implements a “find” command to search for documents with the specified data. If the criteria is empty, an exception is raised as you cannot read empty data.
9. The **ReadAll** method follows a query-like structure. This method is used to retrieve all data to populate the data table used later on in development. Be sure to structure this query like the following to avoid issues retrieving all data: “database.collection.find(data, {"\_id" : False})”
10. The **Update** method accepts two parameters and implements the “update” command from MongoDB. The first parameter queries for the document(s) to be updated, the second parameter holds the updated information for the existing document queried. If the parameters are empty or if the update is unsuccessful, an exception is raised.
11. The **Delete** method utilizes the “remove” command within MongoDB. This function takes a query that matches the document(s) to be deleted. If the query is empty or if the deletion is unsuccessful, an exception is raised.
12. To finish the backend component of the application, we must test it. In order to test your new CRUD class, tests should be written to verify each CRUD methods’ functionality. Using a software such as Jupyter Notebook, the developer should import the CRUD class they created, as well as unittest and the MongoClient to develop unit tests for each method.
13. Next we move onto the frontend component of the application, within Jupyter Notebook, instantiate the CRUD class you developed, import the Dash frameworks we intend to use, and Pymongo.
14. Authenticate the user, then move on to developing the layout of the dashboard. Begin with importing the file used for the client’s logo and create a header which incorporates said logo along with a title, and a unique identifier.
15. Next list out the Radio Items to be used as filters for the data. This is done using Dash Core Components and should include Water Rescue, Mountain Rescue, Disaster Rescue, and Reset Data.
16. Now create an interactive data table which houses the data from all three animal shelters. To do so, utilize the ReadAll method from your CRUD class.
17. Using Dash Core Components, create both a pie chart and a geolocation chart. This is done through a series of callbacks which connect to the data table itself. This allows the widgets to be updated when the data table is updated when a filter is applied.
18. Lastly, ensure the radio items work as filters by developing queries based on the client’s specifications. Each radio item filters the data table and widgets based on the developed queries and is connected to MongoDB using the Read method within the CRUD class.

## Challenges and Solutions

One of the main challenges faced when developing this application was retrieving all of the data from the MongoDB dataset to populate the Dash data table. I would attempt to run a retrieve all query but the result from said query was not correctly parsed by Dash’s framework. This resulted in a constant loading screen with no data loaded. In order to solve this problem, I had to add the projection syntax to my CRUD ReadAll method to not send “\_id” values. Within my CRUD class, this fix looks like “database.collection.find(data, {"\_id" : False})” in the ReadAll method.

## Contact

Sarah Deleppo

sarah.deleppo@snhu.edu