# CS 340 Grazioso Salvare Dashboard README

**About the Project/Project Title**

*Grazioso Salvare Dashboard –* This dashboard operates as a user interface to interact with data stored in a MongoDB database using a CRUD middleware module implemented in Python. The dashboard facilitates a web interface to view the records retrieved based on filtering and interactions with the widgets and can be adapted to present data for a variety of different datasets.

## Portfolio Assignment Questions

How do you write programs that are maintainable, readable, and adaptable? Especially consider your work on the CRUD Python module from Project One, which you used to connect the dashboard widgets to the database in Project Two. What were the advantages of working in this way? How else could you use this CRUD Python module in the future?

When writing programs, it is important to write clean, easily readable and well commented code. These practices combined with breaking large features of projects down into individual functional modules allows creating code that is easily maintainable, readable, and adaptable. When writing the CRUD python module for this project the single task was completed with the single module capable of performing MongoDB interactions that I could then use for the second part of the project by simply importing it. I already have thoughts on how I may re-use this module for future projects that I may integrate with a MongoDB database since it is readily available to perform those interactions in Python.

How do you approach a problem as a computer scientist? Consider how you approached the database or dashboard requirements that Grazioso Salvare requested. How did your approach to this project differ from previous assignments in other courses? What techniques or strategies would you use in the future to create databases to meet other client requests?

Approaching problems as a computer science from my perspective involves integrating technology to create the best solutions possible for a client and society. In the instance of this project, a solution was created using a MongoDB database and Python Scripting to create a ‘full-stack’ solution to the Grazioso Salvare client’s needs. This project was unique from some of my previous courses because this was the first time I needed to develop solutions for the project that served both server-side code and a client-side interface which all functioned together to store data in a modern data storage system, MongoDB. In the future I would continue to adapt and improve my proficiency in working with MongoDB since I now know how easy it makes to integrate with code using the JSON object model.

What do computer scientists do, and why does it matter? How would your work on this type of project help a company, like Grazioso Salvare, to do their work better?

Computer scientists work to find the best possible solutions for individual and societal problems by leveraging and advancing existing technologies. This enables us to advance our profession to create re-usable and rapidly employable solutions such as code projects in this instance to meet the needs of the Grazioso Salvare company to interact with their collection of sheltered animal records. The solution I created would allow any of their employees with a computer and a web browser to rapidly gain awareness of the state of their current animals and rapidly make choices to employ them in a new capability they are likely to excel at rather than staying idle in shelters.

## Motivation

The primary driving factor behind this project is to provide a contracted clients desired product of a full stack solution aimed at easily organizing, filtering, and sorting data through a web browser. The dashboard is currently designed for use with a collection of animal records for the contracted client but could be easily adapted along with the middleware CRUD python module and the MongoDB database.

## Getting Started

To get a local instance of the project up and running enter the following commands at a terminal.

1. Start an instance of MongoDB – ‘mongod\_ctl start-noauth’
2. Import a dataset into the local database to interact with – ‘mongoimport --db AAC --collection animals --type=csv --headerline --file= /usr/local/datasets/aac\_shelter\_outcomes.csv --port 5247’ #Replace the port with your local instance port and the specified db and collection as desired for your data
3. Start the mongo command line interface – ‘mongo’
4. Create a user to connect to the database (AAC in this example) and interact with the data -
   1. ‘use admin’
   2. ‘db.createUser({user: “aacuser”, pwd: passwordPrompt(), roles: [{role: “readWrite”, db “AAC”}]})’
   3. Enter the password for the new user at the prompt.
5. Copy an authentication configuration for the mongod service to the current configuration – ‘sudo cp /etc/mongod\_withauth.conf /etc/mongod.conf’
6. Restart the mongod service to run with authentication required – ‘sudo systemctl restart mongod.service’
7. Change to the project folder and modify Project2.ipynb to include the credentials for the user account created in MongoDB.
8. Run the Project2.ipnyb script in Jupyter notebook to initiate the Python Dash web server.
9. The web server will listen on localhost port 8050 by default which can then be accessed by navigating to <http://127.0.0.1:8050/> in a new browser tab.

This project was created using Python technologies that functioned as both a client interface and middleware to interact with MongoDB. It was challenging to determine how certain errors were being encountered based on data formatting from either the Python middleware or the underlying MongoDB dataset. At runtime there may be minor errors encountered based on how data is being accessed by the incorporated Python modules for each of the widgets but none of this impacts the functionality of the dashboard for the intended use. One other significant challenge when completing this project was properly formatting the HTML elements within the dashboard using the integrated Dash syntax. A lot of trial and error was performed to create the solution for this project; however, I intend to pursue future improvements to make the dashboard more functional and efficient.

## Installation

This project is written to be run with Python3.6 or later and relies upon the following Python modules being installed on the system where it is running: pymongo is a Python module that facilitates interactions with a MongoDB database as a client, bson for binary encoding of JSON objects, getpass for retrieving an input password, pandas for formatting data from the database into dataframes, dash, dash\_leaflet, dash\_core\_components, dash\_html\_components, dash.dependencies, and dash\_table for formatting the web interface and html elements for the dashboard, plotly to create the graph used in the dashboard widgets, base64 for base64 encoding the image file used to render the logo within the dashboard, and urllib for parsing the password entered to encode special characters.

The dashboard/webserver Project2.ipynb is intended to be opened and run with Jupyter Notebook.

Additionally, this project relies on connecting to an existing MongoDB instance that is either locally or remotely accessible for accessing and updating data. MongoDB was chosen for the flexibility in scaling, rapid response to queries and ease of integration with Python

## Usage

An instance of MongoDB must be running with imported data following the steps presented above. The Project2.ipynb should be run within Jupyter notebook within the project folder after modification to use the correct MongoDB user credentials and database/collection specification. After starting the script, the dashboard can be access by navigating to <http://127.0.0.1:8050/> using a modern web browser. The dashboard will automatically populate a table with unfiltered data retrieved from the MongoDB database. Additionally, below the table two widgets will display pie graph representation of breeds presented within the current page of the table and a map presenting the location of the first animal in the table which will both update dynamically as new data is retrieved or filtered. The data presented in the table can be filtered using two methods. First there is a row of four buttons at the top of the table which will apply three filters to present records matching qualities Grazioso Salvare has deemed desirable for dogs acting as ‘Water Rescue’, ‘Mountain or Wilderness Rescue’, and ‘Disaster or Individual Tracking’. These three different filters can be applied by clicking the corresponding button and can be cleared by clicking the ‘Reset Filters’ button. Additionally, the data presented in the table can be sorted by clicking on the arrows on an associated column and filtered by entering a value in the second gray row of a column and pressing enter to filter records based on values that match in that column. The table accepts multiple filters in this manner allowing granular filtering of the data. To clear these filters just delete the filter value and press enter.

### Code Example

The following code example demonstrates modifying variables within the Project2.ipynb script to provide credentials for accessing the MongoDB instance using a user of ‘aacuser’ and password of ‘Example123!@#’.

‘’’

…

username = “aacuser”

password = “Example123!@#”

database = “AAC”

collection = “animals”

port = “52471”

…

‘’’

\*While this example shows embedding cleartext credentials within the script, a more secure and preferable option would be to enter the password at runtime using a module like getpass.

### Testing

Testing has been performed for each desired interaction detailed within the client Dashboard Specification documentation. This includes applying filters using the included buttons, filtering data within the table using search strings, clearing filters, paging through the table for multiple outputs, sorting, observing the updated content for both the map and graph as data changed within the table, and engaging the html anchor website from the Grazioso Salvare logo. Each of these actions can be seen in the accompanying screenshots that follow. The map and pie graph can be observed changing as different filters are applied throughout the screenshots.

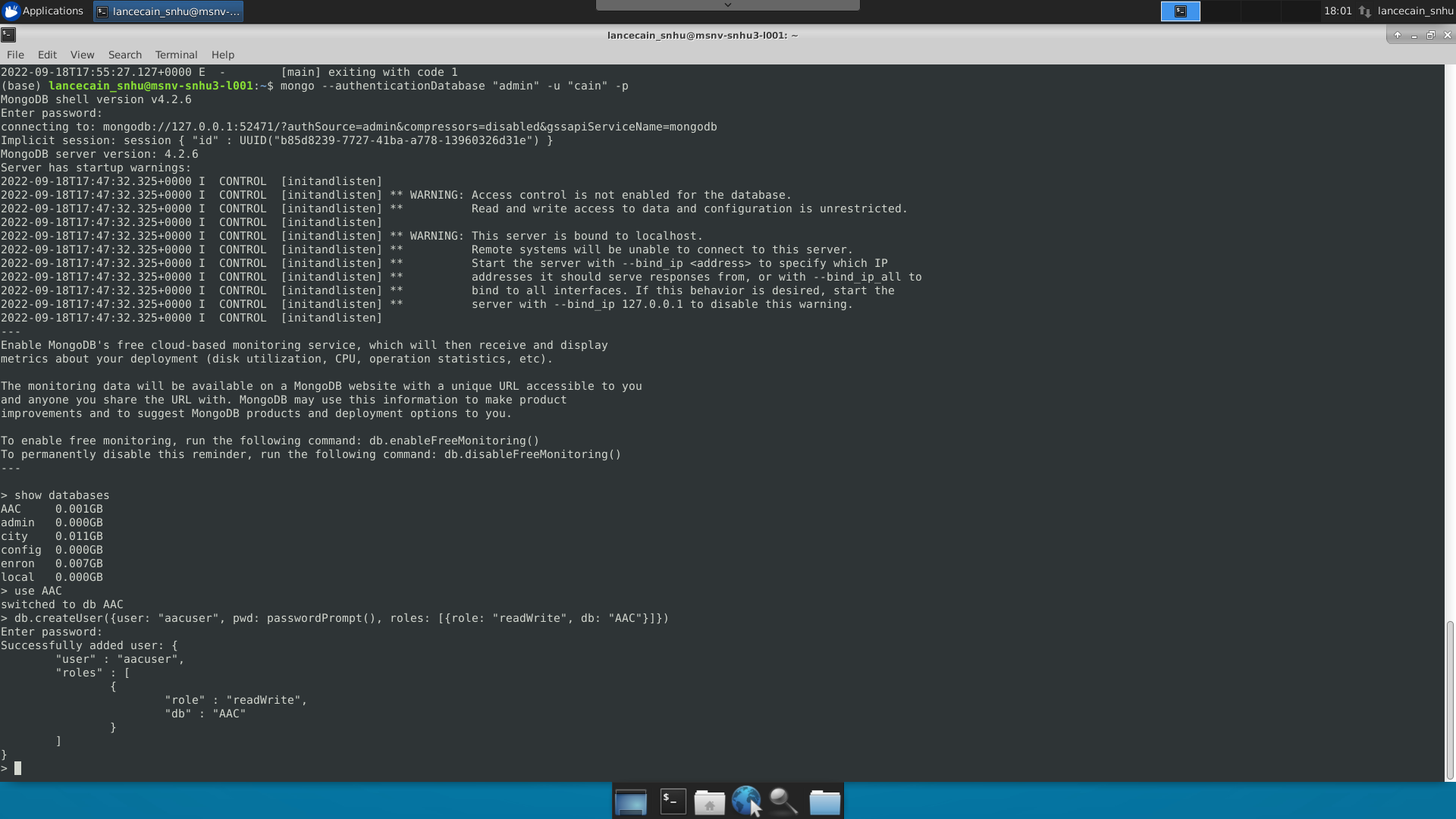
### Screenshots

Importing the Austin Animal Center Outcomes data CSV using the MongoDB import tool.

Graphical user interface

Description automatically generated

Authenticating as an administrator account and “aacuser” user account



Graphical user interface, text

Description automatically generated

Starting MongoDB with user authentication and loaded data

Graphical user interface, text

Description automatically generated

Initiating Project2.ipynb in Jupyter Notebook after modifying the connection variables

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Accessing the dashboard in a new browser tab

Graphical user interface, text, application

Description automatically generated

Opening the Grazioso Salvare logo html anchor in a new tab

Graphical user interface, text, application

Description automatically generated

New tab to [www.snhu.edu](http://www.snhu.edu) opened and view of the dynamic graph and map on the dashboard

Graphical user interface, application

Description automatically generated

View of the data table without any filters

Graphical user interface, text, application

Description automatically generated

Water Rescue Filter Applied

Graphical user interface, application

Description automatically generated

Mountain Rescue Filter Applied

Graphical user interface, application

Description automatically generated

Disaster Rescue Filter Applied

Graphical user interface, application

Description automatically generated

Reset Filters

Graphical user interface, application

Description automatically generated

Filter data viewed in the table by typing in the second row of a column and pressing enter

Graphical user interface, text, application

Description automatically generated

Reset by clearing filter string and pressing enter

Graphical user interface, text, application

Description automatically generated

Sort by column

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Navigating through multiple pages and scrolling to display additional columns

Graphical user interface, application

Description automatically generated

## Additional Resources

This project would not be possible without the helpful insights provided at the following links and project documentation made available for the technologies employed like MongoDB and pymongo. I encourage everyone to review these items to get a better understanding of the inner workings of this project.

https://www.mongodb.com/docs/manual/reference/method/

https://pymongo.readthedocs.io/en/stable/

https://plotly.com/python/pie-charts/

https://stackoverflow.com/questions/72473994/make-column-width-fit-to-data-and-header-in-dash-datatable

https://community.plotly.com/t/clickable-a-tag-on-image-source-in-dash/10016

https://stackoverflow.com/questions/20657951/multiple-regex-using-and-in-mongodb

https://www.statology.org/pandas-unique-values-in-column/

https://www.marsja.se/pandas-count-occurrences-in-column-unique-values/#:~:text=To%20count%20the%20number%20of%20occurrences%20in%20e.g.%20a%20column,in%20the%20column%20%E2%80%9Ccondition%E2%80%9D.

https://lifewithdata.com/2022/02/28/how-to-create-a-pie-chart-in-plotly-python/

## Contact

Lance Cain: [lance.cain@snhu.edu](mailto:lance.cain@snhu.edu)