### CS 340 README: GLOBAL RAIN for GRAZIOSO SALVARE



## About the Project/Project Title

For this project, the software engineering team at Global Rain will work with our client, Grazioso Salvare, to create an application that allows them to search for specific dogs in animal shelters that can be trained for search-and-rescue operations. It is the task of our team to create a database and client-facing web application that users at Grazioso Salvare can use to find the perfect candidates in five animal shelters across the Austin, Texas region for search-and-rescue training.

## Motivation

Grazioso Salvare has taken on the noble mission of training rescue animals, and will need assistance in streamlining the process of finding a perfect candidate to begin training for their crucial missions. The goal is to provide them with a functional, user-friendly application that will allow them to find ideal dogs for training based on their ability to be trained for water rescue, mountain/wilderness rescue, or disaster rescue/individual tracking. An application like this one can save Grazioso Salvare a lot of time and money, replacing processes like calling each animal shelter one-by-one for information with one that provides that information in a matter of seconds.

## Getting Started



In order to access the animal shelter’s database in MongoDB, follow these simple steps:

1. Start the MongoDB server and log in with username and password. Those who do not have a username/password or would prefer not to work with one can use the command “mongod\_ctl start-noauth” to start.
2. Use command “mongo” to start using Mongo
3. The command “show dbs” can be used to provide an overview of the available databases in Mongo. Entering “use *database-name*” will easily switch from one database to another.
4. To set up user accounts and different privileges, please refer to the MongoDB [Manual Enable Access Control](https://www.mongodb.com/docs/v4.2/tutorial/enable-authentication/) tutorial. User authentication can be enabled thereafter with the following commands:

*/bin/cp /etc/mongod\_withauth.conf /etc/mongod.conf*

*/bin/systemctl restart mongod.service*

To disable user authentication, please use the following commands:

/bin/cp /etc/mongod\_noauth.conf /etc/mongod.conf  
/bin/systemctl restart mongod.service

If you need to verify that the user authentication is active and you want to use a new username/password combo that you created, the following command can assist with that. Be sure to had access to your port number (check MongoDB log for port number).

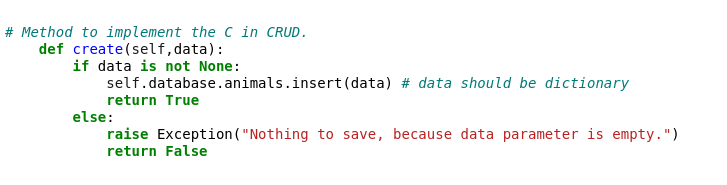
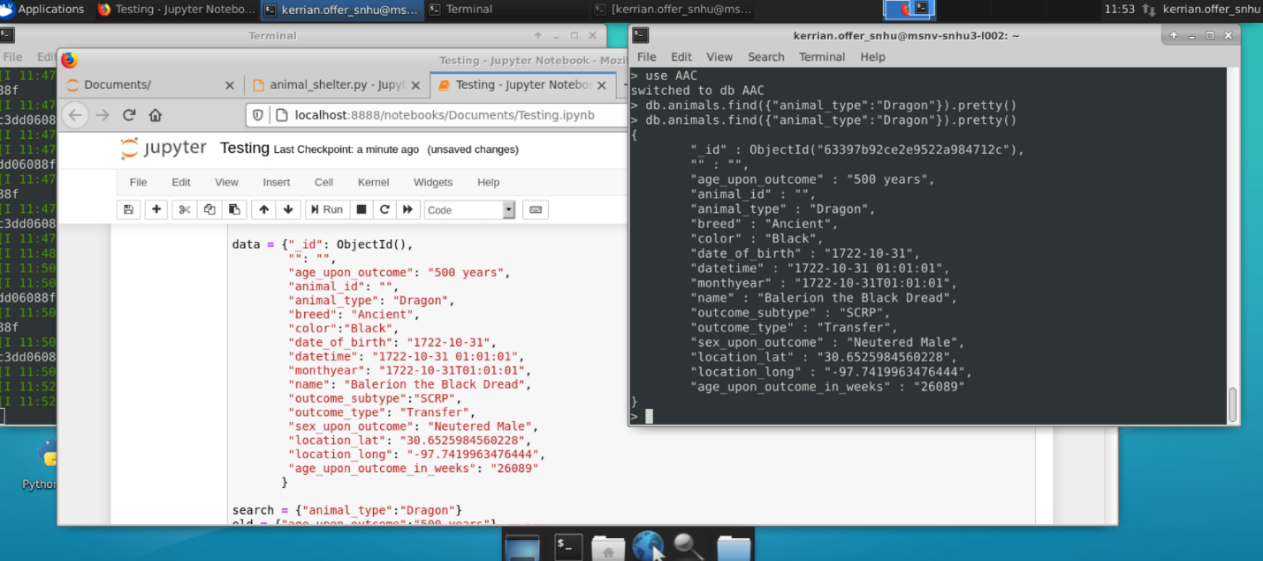
*mongo --port ##### --authenticationDatabase "username" -u "password" -p*

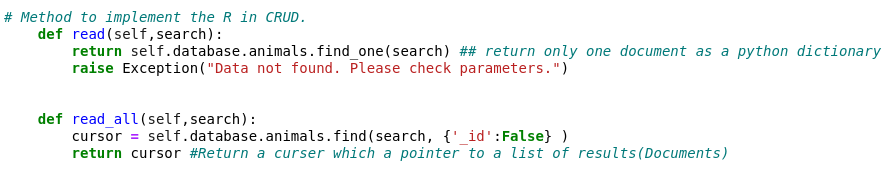
You can always check each database with the db.*database-name*.getUsers() to receive a list of users added to that database and their privileges.

1. Once you and/or your users are able to access MongoDB itself, various [database commands](https://www.mongodb.com/docs/manual/reference/command/) can be used to perform different tasks.

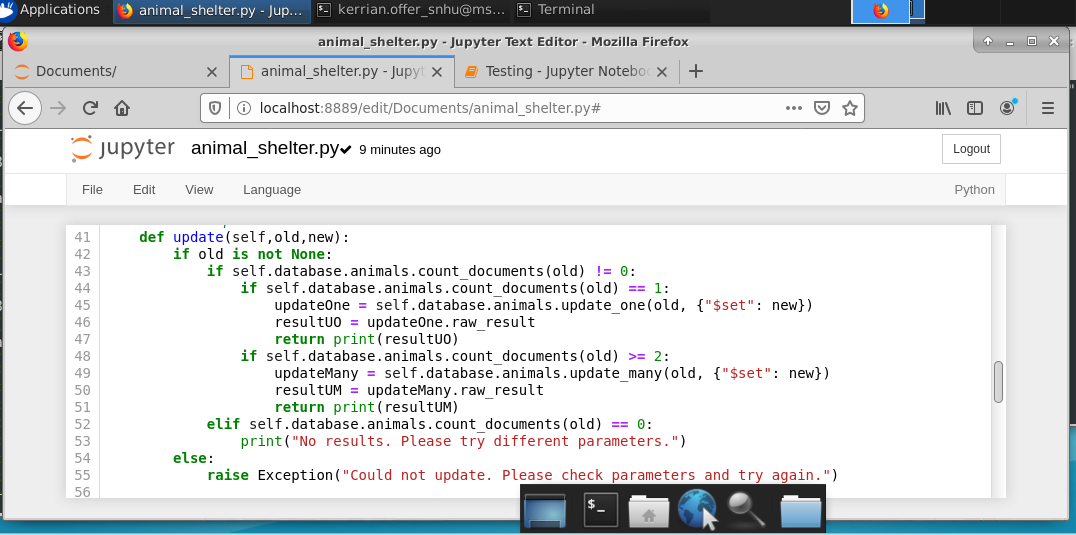


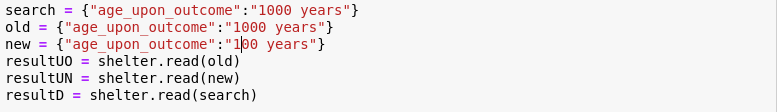
With MongoDB running, Jupyter Notebook should be opened with its own terminal running (right-click and change settings to run the terminal before starting the application). Once Jupyter is open, create a designated folder for your files.

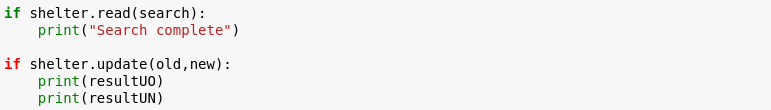
1. In order to utilize the benefits of Python to make changes to the database.
   * We will be utilizing Restful API’s CRUD (Create, Read, Update, Delete) in Python to allow our application to create, read (find), update, and delete entries without having to physically access the database inside MongoDB itself.
   * In order to accomplish this, MongoDB was opened, MongoDB was accessed (without authorization to avoid complications), and then Jupyter was opened with its own terminal in order for all to be running at the same time.
   * From Jupyter, we are able to utilize CRUD to do perform various tasks which can immediately be tested and found in the MongoDB itself.
   * Testing was also done through Jupyter. This software has the capability to make updates to the MongoDB even during testing. It is recommended that a new entry (or entries) is made strictly for testing in the database to avoid making irreversible changes to actual entries during the CRUD process.
2. In MongoDB, there will need to be two files created: a .py file to create the classes that will create, read, update, and delete data (or CRUD), and a .ipynb file to actually run the classes that will make updates to the database in MongoDB.
   * In the .py file, use ‘from pymongo import MongoClient’ to establish a connection between the python file and MongoDB. From this file, log ins and database access can be accomplished by using CRUD methods to make changes without having to manually do them in MongoDB.
   * In the .ipynb file, the log in credentials can be used to access a database with user authentication enable. It is also here where data fields can be written and then tested with different calls to create, find, update, and delete documents. Creating the .ipynb file as a Python 3 file instead of a text file allowed not only instant test results, but also eliminated technical issues I ran into when using a text .py file that was later converted to a .ipynb file. This approach is not mandatory but recommended.
   * .ipynb files can also be used to build the application that users will utilize to find information within the database. Using imports like dash, plotly, and pandas will allow the use of information in the database to be represented with visually with the use of buttons, radio buttons, dropdown menus, graphs, embedded spreadsheets, etc.
3. **Create**: In order to create new data (or documents), a class can be created to not only insert the data into the database but to also check the data that is being sent for missing parameters. If essential parameters are missing, users can be alerted from this class with an exception message and the attempt can be rejected. An example is provided below where an animal (Balerion the Dragon) is created and then found within the mongo database. This screenshot also covers ‘Read’—or the ‘R’ in ‘CRUD’.  
   
4. **Read:** In order to read data within a database, a class can be created to search the database for specific entries based on the provided criteria. The class created for read can handle the search request before notifying users whether their search returned results, returned nothing, or if parameters entered are invalid.

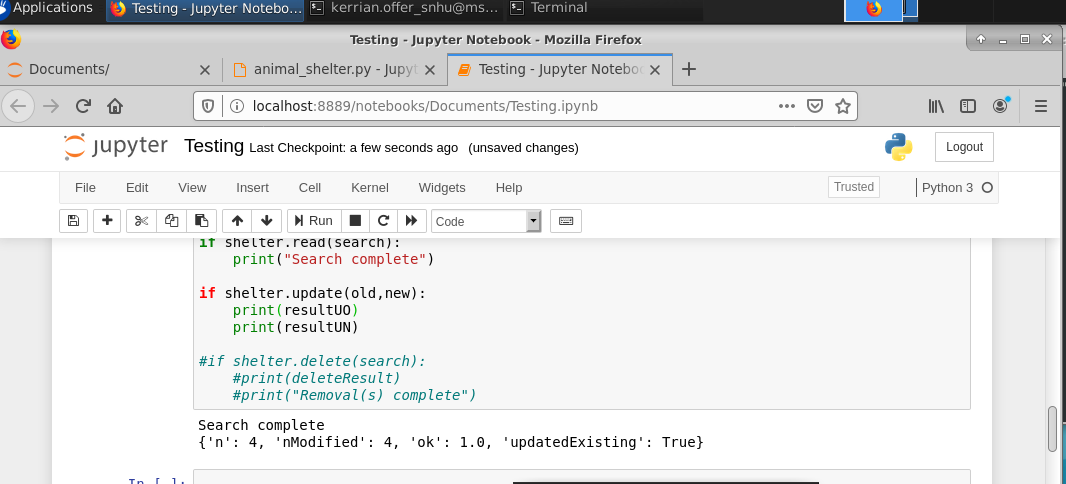


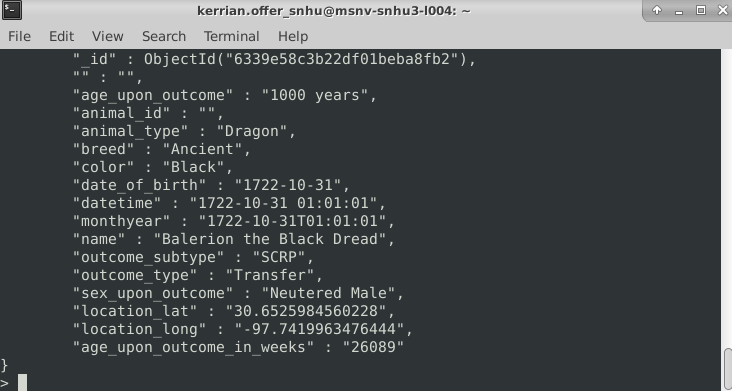
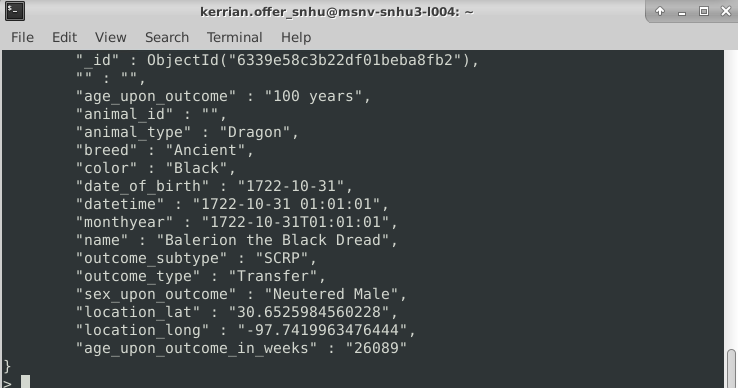
1. **Update:** Updating existing data within the database will require a variable to look for the item of the search and a variable that represents the change that will be made. As with ‘Create’ and ‘Read’, we want to make sure that the parameters are not empty first. If it is empty then an exception is raised to let the user know that an update could not be done. If the search parameter is acceptable, though, then we begin by using the ‘count documents’ feature to check if the search result came back with anything. If the count is anything besides zero then it is considered a success, and the variable of change will be used to update the parameter(s) in question.
   * In this example, our sample animal Balerion—who we added in the ‘Create’ section—will have his age updated from 500 years to 1000 years using the update feature. The result can be returned in JSON format which is used in MongoDB :



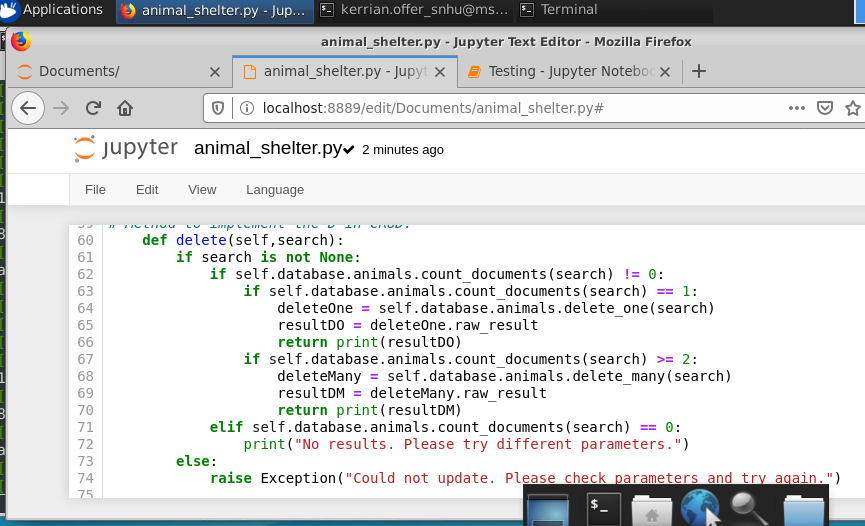


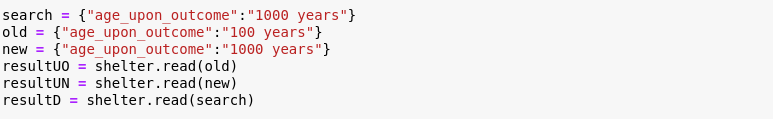


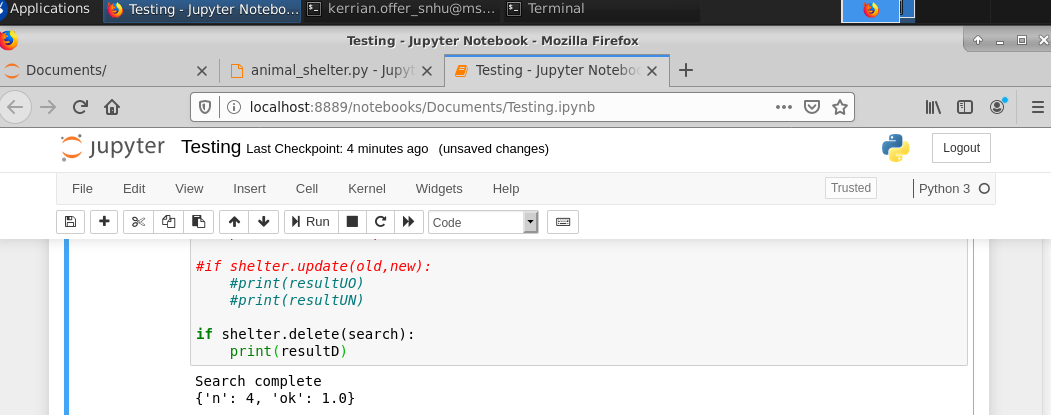


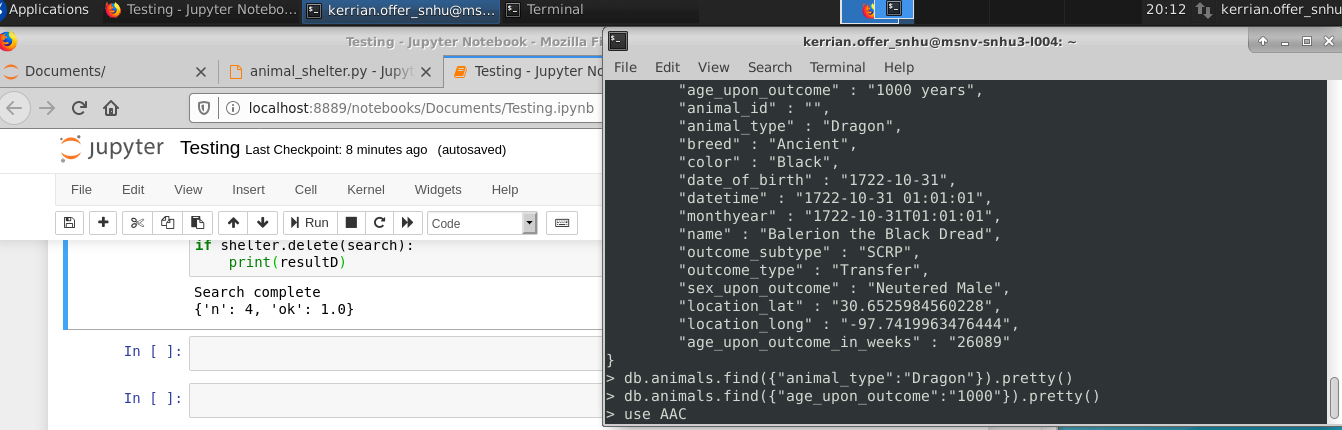


1. **Delete:** The final method behaves similarly to the update method where a variable is used as a query to look through the documents in the database/collection. If that result returns zero search results then the user is notified. The user is also notified if no search was done at all due to a lack of parameters being provided. In order for a successful search to happen, the count of the search result has to be anything besides zero. Upon finding the expected result based on the query provided, a result is deleted as commanded.
   * In the below example, Balerion is deleted from the database and attempting to find him again returns no results.

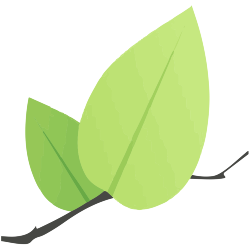






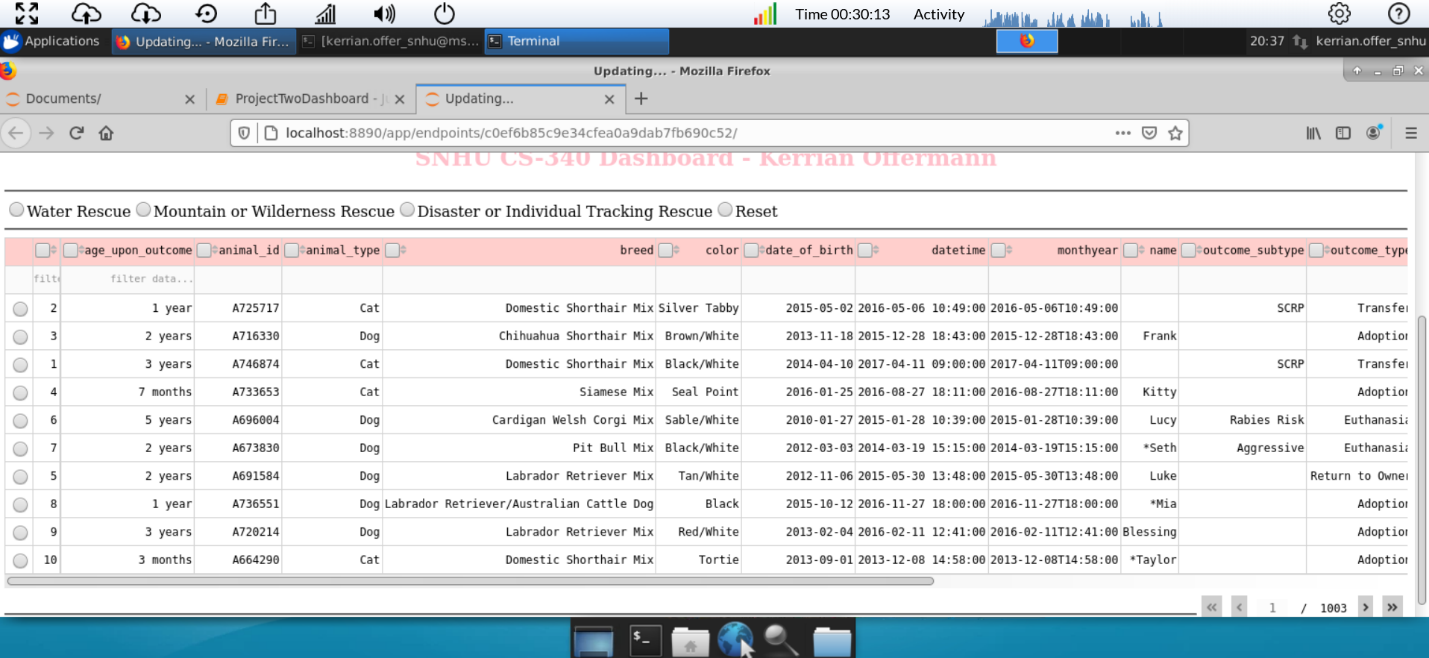
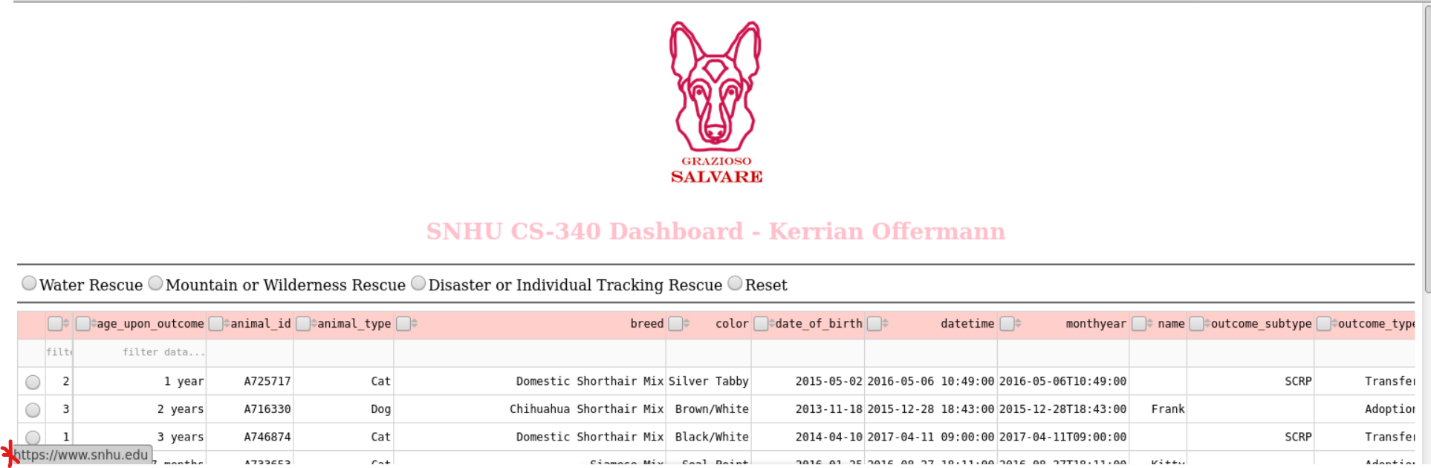
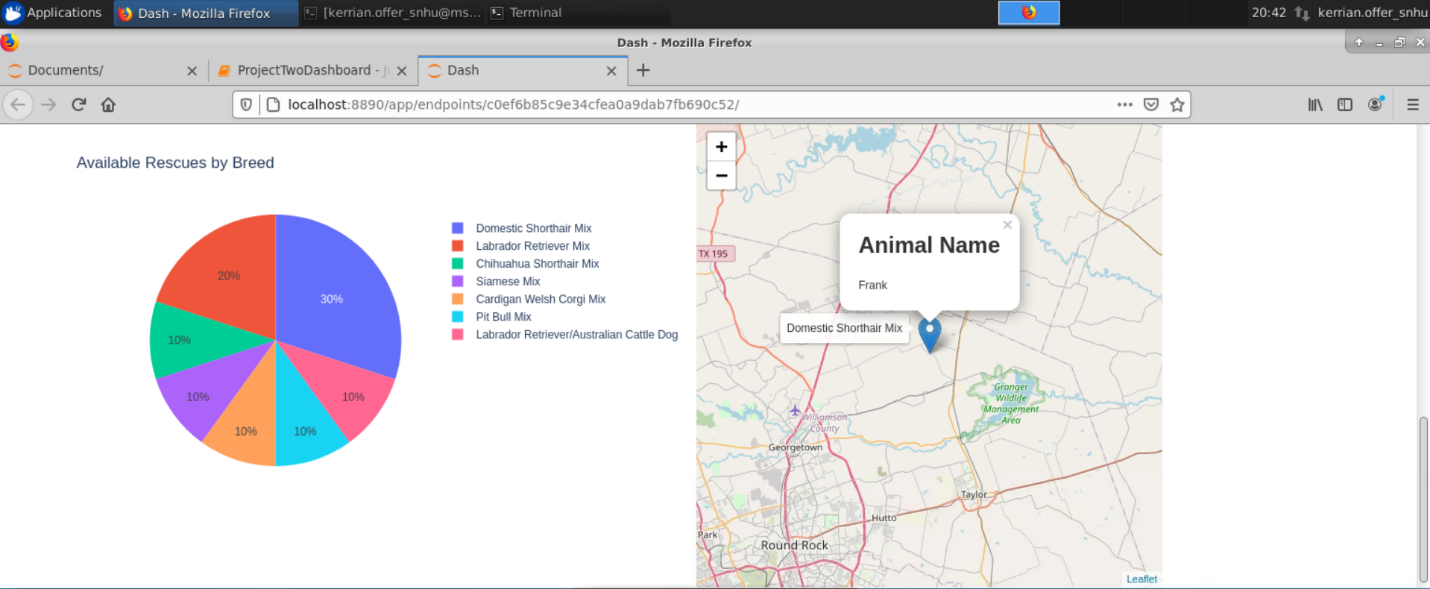


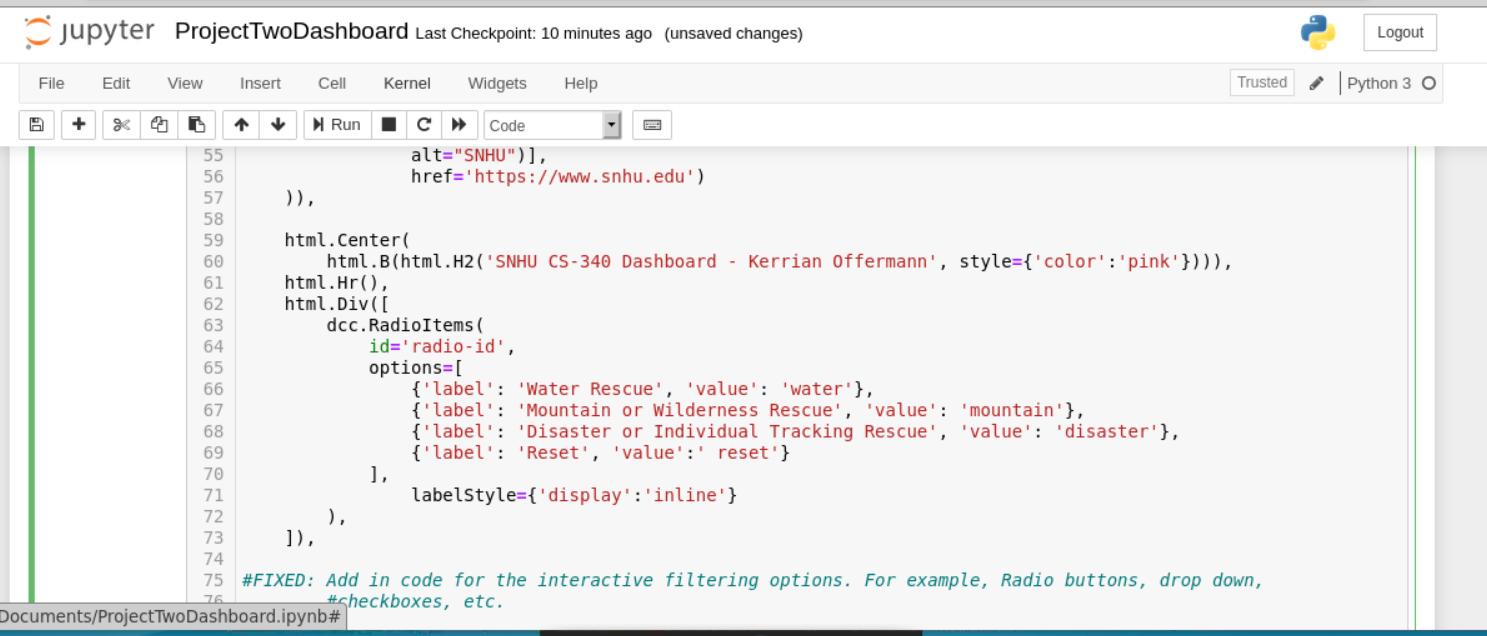
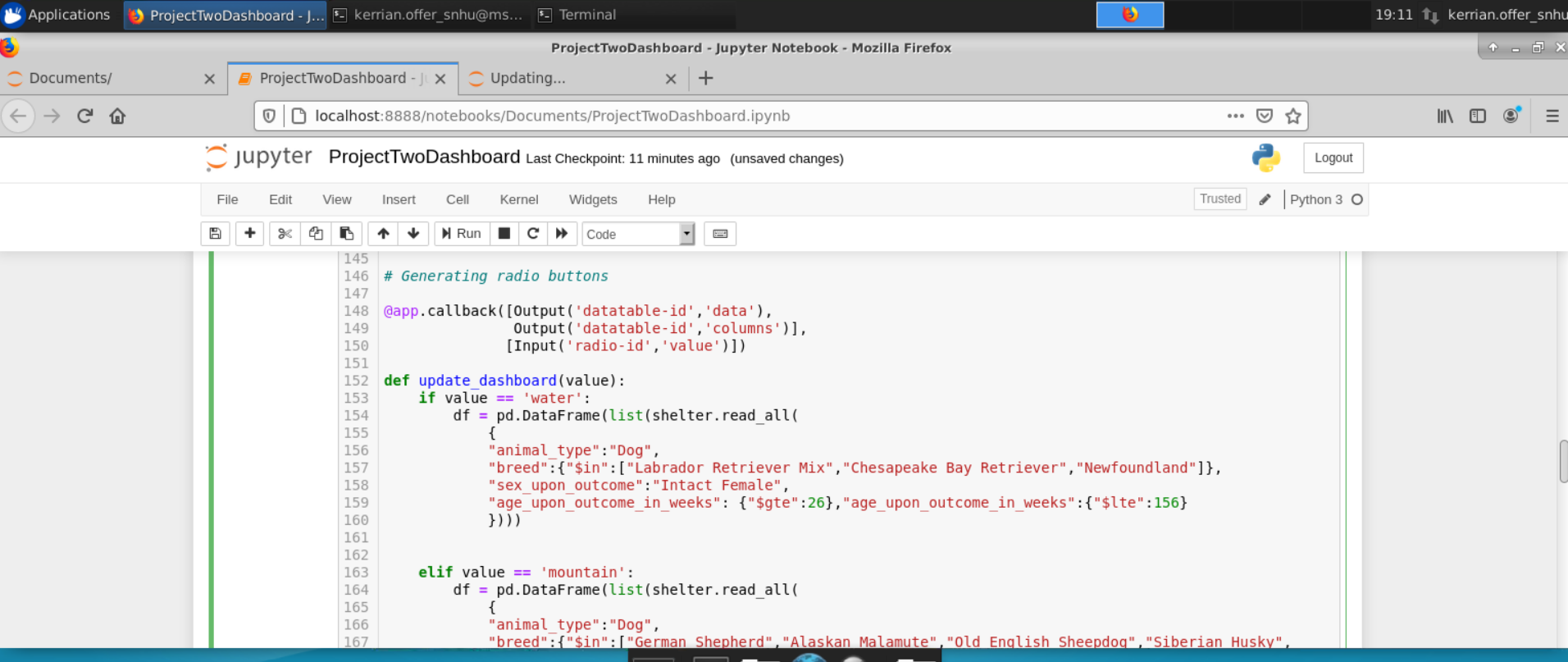
1. Once the Restful API is setup and testing has confirmed that data can be created, read, updated, and deleted from Jupyter, the next step is to use an import like plotly/dash to begin creating a user-friendly method of interacting with the information in the database:

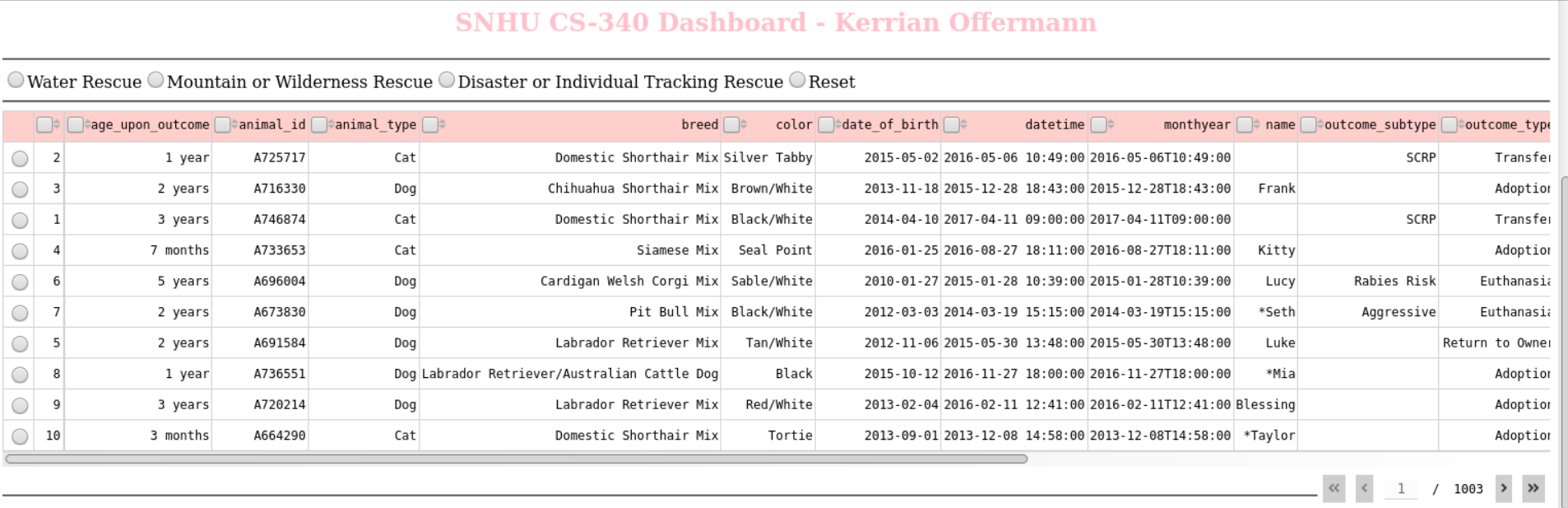
 

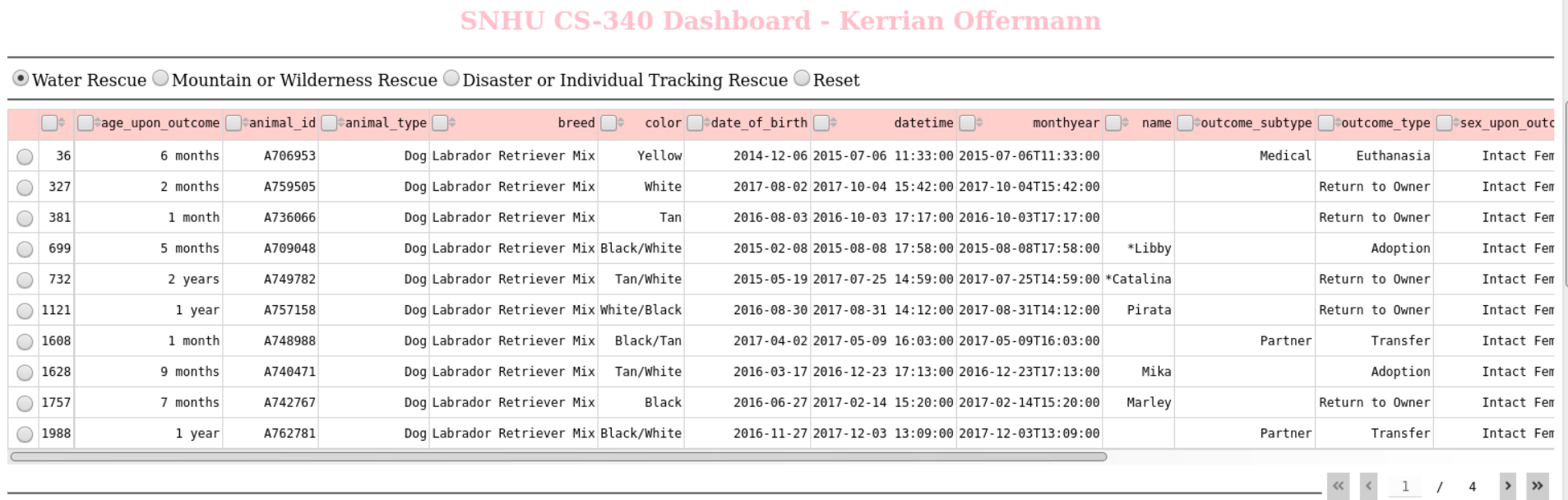
1. Plotly/Dash is a library that allows the use of charts, menus, tables, and other interactive application objects to interact with a database. This user-friendly route to interact with a database removes the need to use complex code to find and alter information, and instead replaces them with simple, clickable interactions that perform the same tasks. Plotly/Dash also allows the use of graphs and maps (via Dash Leaflet) to take data and visualize them.



1. The coding itself is done in Jupyter Notebook using a .ipynb file. In this file, the libraries listed above can be imported in order to create the spreadsheet, chart, and map that is shown above. These visual aids draw data from the database using the CRUD methods created earlier (which is also imported along with the other libraries).
2. After importing the different libraries, the next step is to utilize HTML to set up the layout of the dashboard, and set up the attributes and features that will be setup in the next section using callbacks. App callbacks install which part of the database needs to be used for the outputs and inputs of each section of the database. For example, when setting up our radio button menu we first need to establish the menu will look like on the dashboard.  
   
3. We then use app callbacks to set up the radio menu to populate different filters based on the unique criteria specified by Grazioso Salvare. In this example, we set up the app callback to look at our database for the information we need, and what happens to that data when an input is made (the output is essentially what will be returned). If the user selects the water rescue option, the code below will look into the database, follow the ‘read’ restful API that we coded in our .py sheet, return information using the commands we coded with the criteria listed below.:  
   
4. Doing this creates a radio button menu where each button searches our database with the query coded into the app callback section and returns it with each press instead of requiring users to do manual searches in MongoDB. Here is a comparison between an unfiltered view and then one where we select ‘Water Rescue’ from the radio buttons:





## Installation

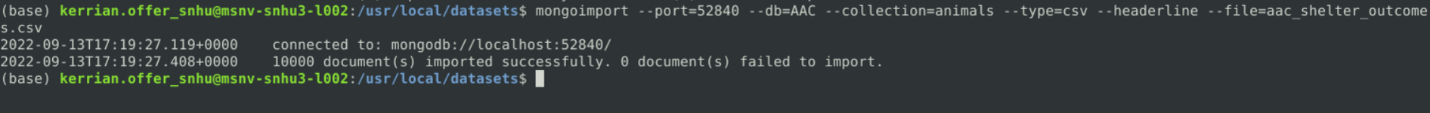
The tools needed to are as follows:

* MongoDB: The MongoDB application will allow us to perform a wide array of database-related tasks such as adding users, establishing authentication, and creating, updating, or removing entries within imported databases.
  + Install: https://www.mongodb.com/docs/manual/installation/
  + Documentation: https://www.mongodb.com/docs/
* Python: Python is a programming language that will be used to perform some of the tasks that MongoDB has with more speed and flexibility.
  + Install: https://www.python.org/downloads/
  + Documentation: https://www.python.org/doc/
* Jupyter Notebooks: This program uses Python codes to organize, create, and update information in MongoDB with a user-friendly application. Changes made to the database using Python codes in Jupyter and instantly updated in MongoDB.
  + Install: https://jupyter.org/install
  + Documentation: <https://docs.jupyter.org/en/latest/>
* Plotly/Dash: As a library, plotly can be imported into programs to offer graphing and application design that users can interact with.
  + Documentation: <https://plotly.com/graphing-libraries/>
* Dash Leaflet: A library that allows the use of interactive maps:
  + Documentation: https://dash-leaflet.herokuapp.com/#start

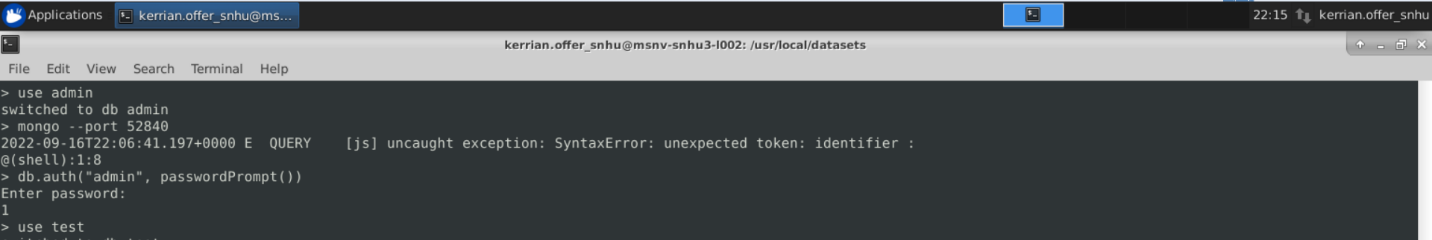
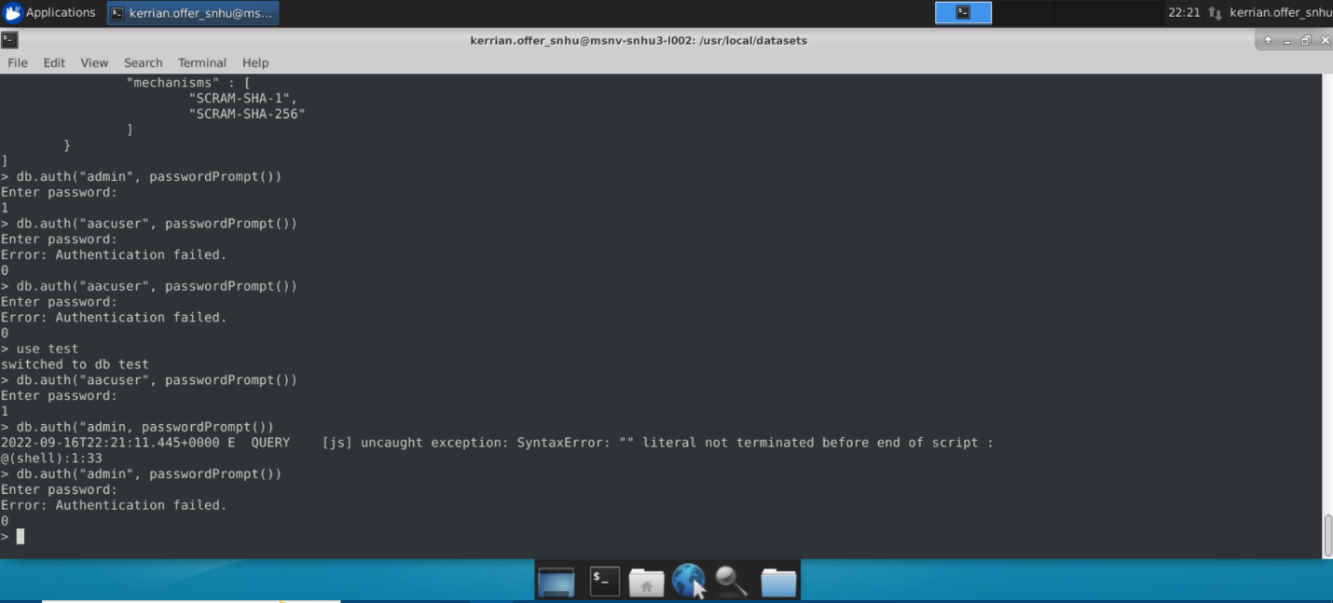
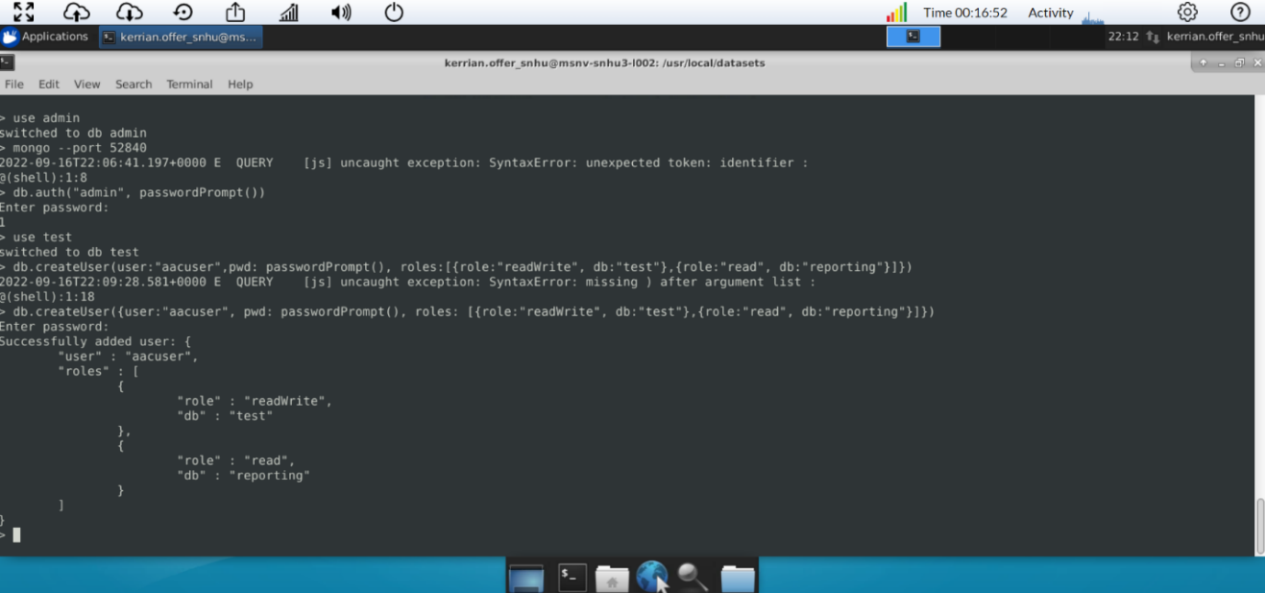
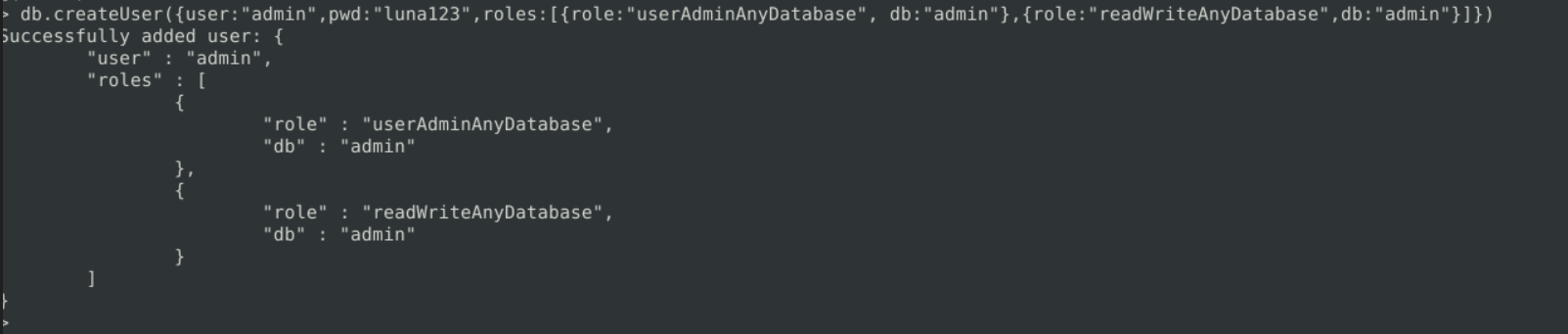
## Usage

### Code Example

MongoDB is used to import database information and then utilize different commands to research or modify the imported information. This can be accomplished with a few pieces of data such as the database name, port number, document type, etc.

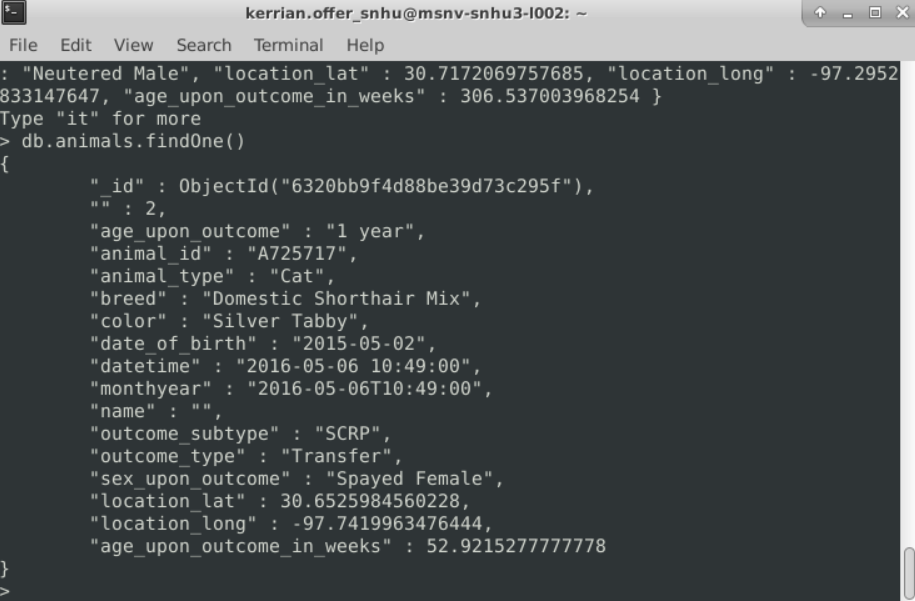


It is possible to log into the different databases using administrative and user accounts. The process of creating them involves a few short lines of code, and then an easy way to log in and out of the accounts when needed.

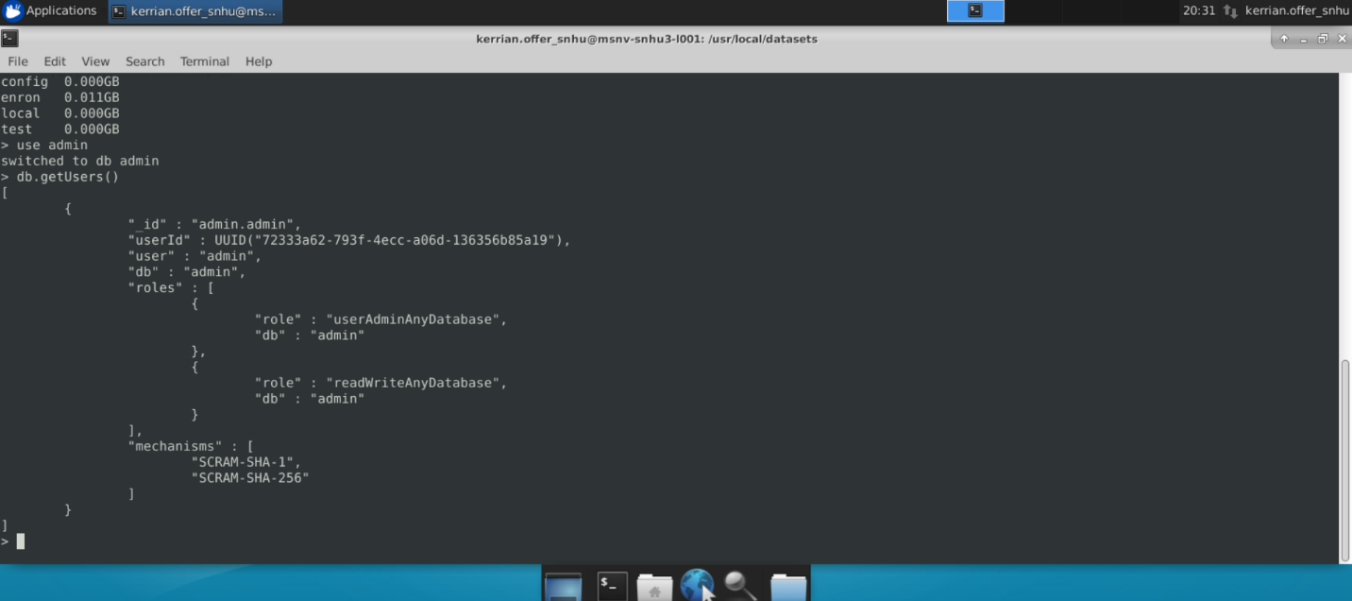


### Tests

In order to confirm that an operation performed as it was meant to do, commands such as ‘find’, ‘findOne’ or ‘getUsers’ allow us to check within a database for a single or multiple entries. For example, choosing the desired database and using the ‘findOne’ command will bring us an entry from the database.



Here is a similar example with users available in the database along with their privileges:



## Roadmap/Features

Here are some features and known issues that we are keeping in mind as the database is being developed into an application:

* Future Features: More charts and graphs will be considered in order to accurately analyze rescue animals that are available within the Austin, Texas area.
* Known Issue #1: Unless I run the .ipynb file as a Python 3 file first in Jupyter, it will return an error that reads “Error loading notebook… Notebook does not appear to be JSON…”. Although Python 3 makes testing a lot smoother, this error should be looked into just in case.
* Known Issue #2: Although the testing file (Testing.ipynb) is running correctly and passing CRUD commands to the MongoDB, there is an issue with ‘IF’ statement for testing updates showing up as red instead of green like the if statements for Create, Read, and Delete. I am looking into this to understand why it functions correctly yet appears red like an error.
* Known Issue #3: Some features may not appear to be functioning due to the long loading time of Jupyter Notebook when running dash application. It is advised that you wait until the page no longer says ‘Updating…’ to ensure if a feature is missing or still loading. Solutions to speed up the load time are being looked into.

## Contact

Your name: Kerrian Offermann [kerrian.offermann@snhu.edu; koffermann@globalrain.org]