# CS 340 README Template

**Company Information**

*Grazioso Salvare is a business that looks for dogs that would be good for search and rescue training. They have a list of specific types of dogs that are good for certain conditions, as well and preferred sex. These specifications help them find the perfect dogs to train for search and rescue. Grazioso Salvare is looking for a software application that can work with their existing data in order to categorize and identify dogs that would be a good fit. They also wish for the data to be presented in an interactive dashboard that lets them filter and query in a user-friendly way based on their specifications.*

**Project Synopsis**

*This project seeks to bolster the query process in a Mongo database for the company Grazioso Salvare. Through executing Python script and running the Mongo server, CRUD operations will be enabled for streamlined accessibility by end users.*

## Motivation

*Expand and improve upon the query process in customer's Mongo database. By executing Python scripts on a Mongo server, this initiative aims to make it easier for end users by facilitating various Create-Read-Update-Delete (CRUD) operations with ease.*

**Key Issues**

*Some key pieces that Grazioso Salvare would like to address are filter options (buttons or dropdowns) based on rescue type breeds, a data table that dynamically updates based on the given filter, and a geolocation chart with another chart that dynamically responds to filtering options.*

*The GS organization has been given the opportunity to train service animals for local law enforcement agencies. However, all their locations must be able to share records appropriately, adding, deleting, and modifying them as needed; as well as a dynamic way to read records with certain specifications to easily identify which animals are appropriate to each type of work:*

* *Water rescue (looking for intact female dogs, between 26 and 156 weeks old, that are either Lab mixes, Chesapeake Bay Retrievers or Newfoundlands);*
* *Mountain or Wilderness Rescue (looking for intact male dogs, between 26 and 156 weeks old, that are either German Shepherds, Alaskan Malamutes, Old English Sheepdogs, Siberian Huskies or Rottweilers);*
* *Disaster or Rescue tracking (looking for intact male dogs, between 20 and 300 weeks old, that are either German Shepherds, Doberman Pinscher, Golden Retriever, Bloodhound or Rottweilers).*

**Solution**

*Our team, together with Grazioso determined that a web-based application is the most advantageous solution, and our team is in charge of incorporating diagrams and maps to help improve their visualization of data - aiding those partaking in searches with their choices. Additionally, they require a script which can execute CRUD operations to manipulate data in their database, as well as an adjustable algorithm to make animal queries more appropriate according to each rescue type. The goal is to put into place an adaptable repository solution that enables their own team to keep records up-to-date across all countries where the company is active.*

## Tools Used to Complete the Project

*To begin the development of such functionality, a data set was imported as the starting point before creating indexes to adding CRUD functionalities. Testing can be implemented in the next phase to ensure its successful operation. Ultimately, this endeavor aims to facilitate access of valuable information stored within a mongo environment.*

***Installation***

*1. Command Line;*

*2. Pymongo (check if version is up-to-date by using pip install pymongo in the command line;*

*3. Jupyter Notebook for automated testing.*

***Step-by-Step***

* *To get started, open Mongo and use the mongoimport command to upload the aac\_shelter\_outcome.csv file into the AAC database and animals collection. Once this is complete, verify the documents were added correctly by checking for newly created databases and collections.*
* *After successful verification of uploaded data, create simple and compound indexes for the documents in the AAC database for faster retrieval of information.*
* *To ensure secure access to this collection, create an admin account as well as a user account with readWrite permissions on the documents within AAC.animals. This will grant users restricted access to the collection while still allowing them to make changes when necessary.*
* *To use CRUD operations on the documents in the AAC database, create both ipynb and py files to perform the necessary operations. This will allow for efficient access and manipulation of documents in the collection.*

**Design**

*In order to write programs that are maintainable, readable, and adaptable, it is crucial to insert comments when necessary in order to outline what each process does. In this project, it was important written comments for each different function in both the crud module and the application file because each piece was responsible for something specific and without comments, it is very easy to get lost in the code. It is important methods and calls properly in order to clarify how they operate.*

***[CRUD](/animal\_shelter.py)***

*In addition to outlining CRUD functions, the Python module ensures proper user access to database by taking in username and password data for authentication. It also supports handling for the following common user errors:*

| **Phase** | **Error** | **Reason** |
| --- | --- | --- |
| CREATE | Nothing to save, because data parameter is empty | Required information was not entered in create() |
| READ | Nothing to save, parameter is empty | read() function did not return any results |
| UPDATE | Nothing to save, parameter is empty | User did not enter the key or value to execute update(). |
| UPDATE | Nothing to save, parameter not type dict | Value specified does not match the database data type. |
| DROP | Unable to delete entry | drop() returned false. Specified record might not exist. |

### Code Example

*Example using Create functionality:*

from pymongo import MongoClient

class AnimalShelter(object):

   """ CRUD operations for Animal collection in MongoDB """

   def \_\_init\_\_(self, username=None, password=None):

       # Initializing the MongoClient. This helps to

       # access the MongoDB databases and collections.

       self.client = MongoClient('mongodb://@localhost:27017')

       # self.client = MongoClient('mongodb://%s:%s@localhost:27017/?authMechanism=DEFAULT&authSource=AAC'%(username, password))

       # where xxxx is your unique port number

       self.database = self.client['AAC']

# Complete this create method to implement the C in CRUD.

   def create(self, data):

       if data is not None:

           insert = self.database.animals.insert(data)

           print("Task Finished")# data should be dictionary

           if insert !=0:

               return True

       else:

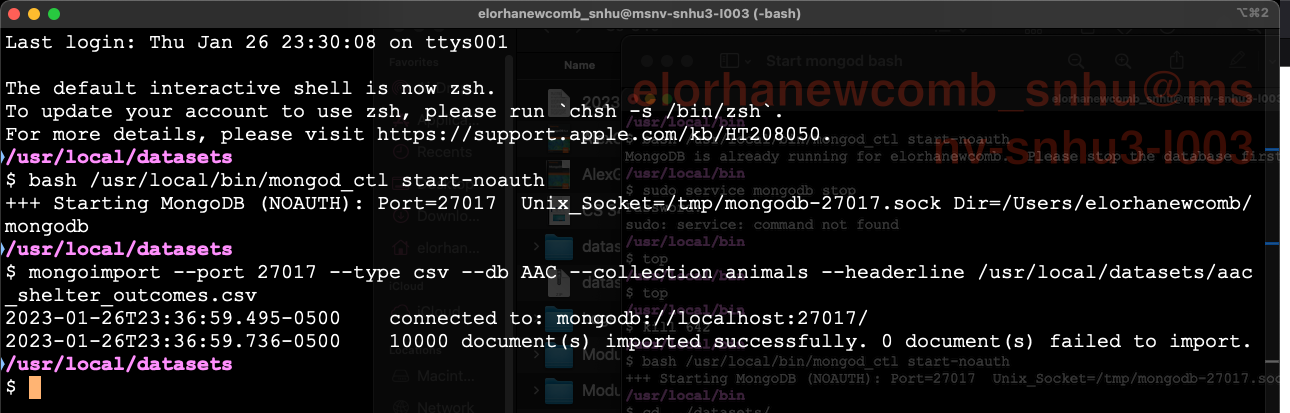
           raise Exception("Nothing to save, because data parameter is empty")

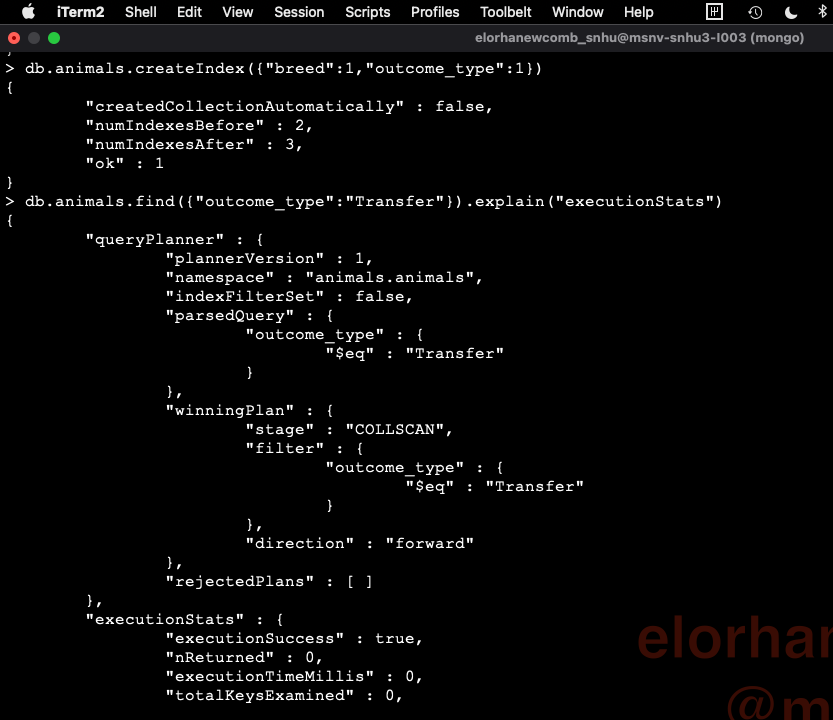
### Tests

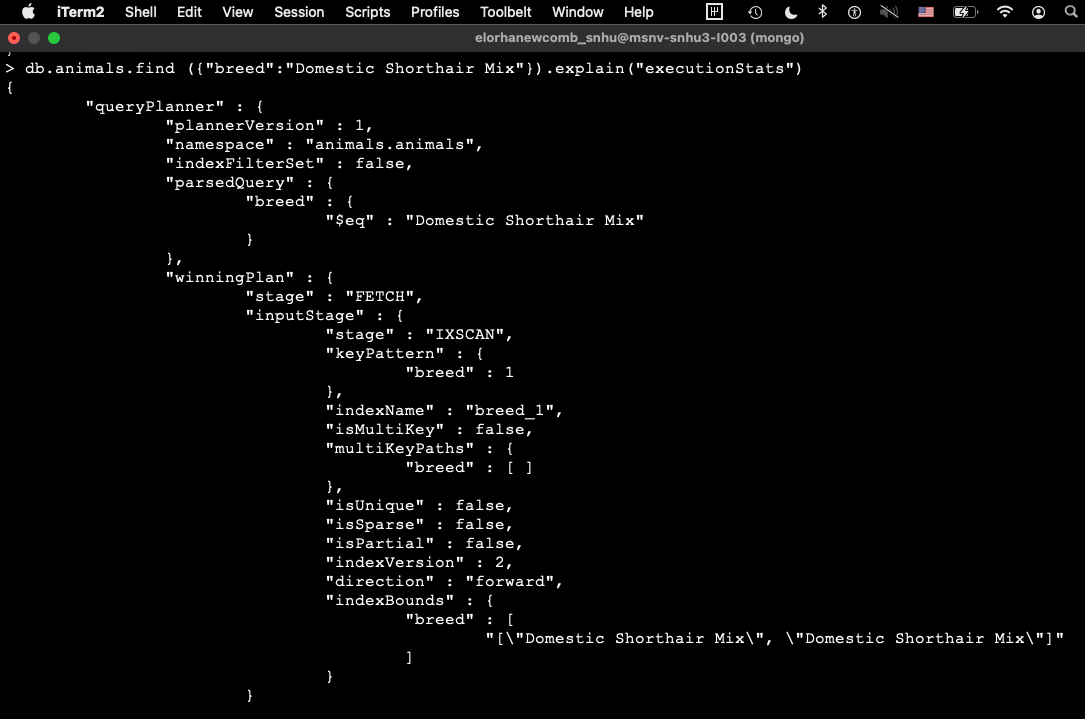
*To assess the anticipated CRUD operations, I incorporated Jupyter notebook software and provided the script created with a set of sample data to be implemented within MongoDB. To validate these actions, I tested crucial areas of the code in the Python script document to be executed one by one, as depicted in screenshots below.*

### Screenshots

1. Upload Animal Center Outcomes into MongoDB importing a CSV file:



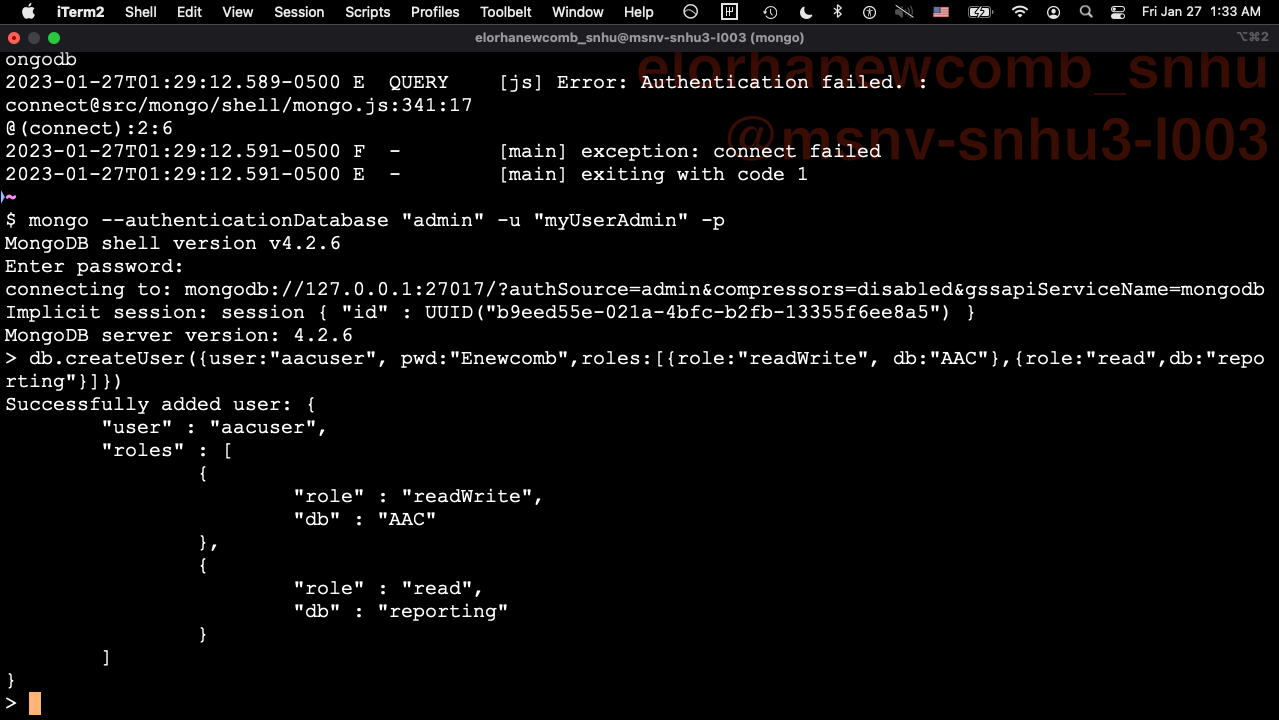


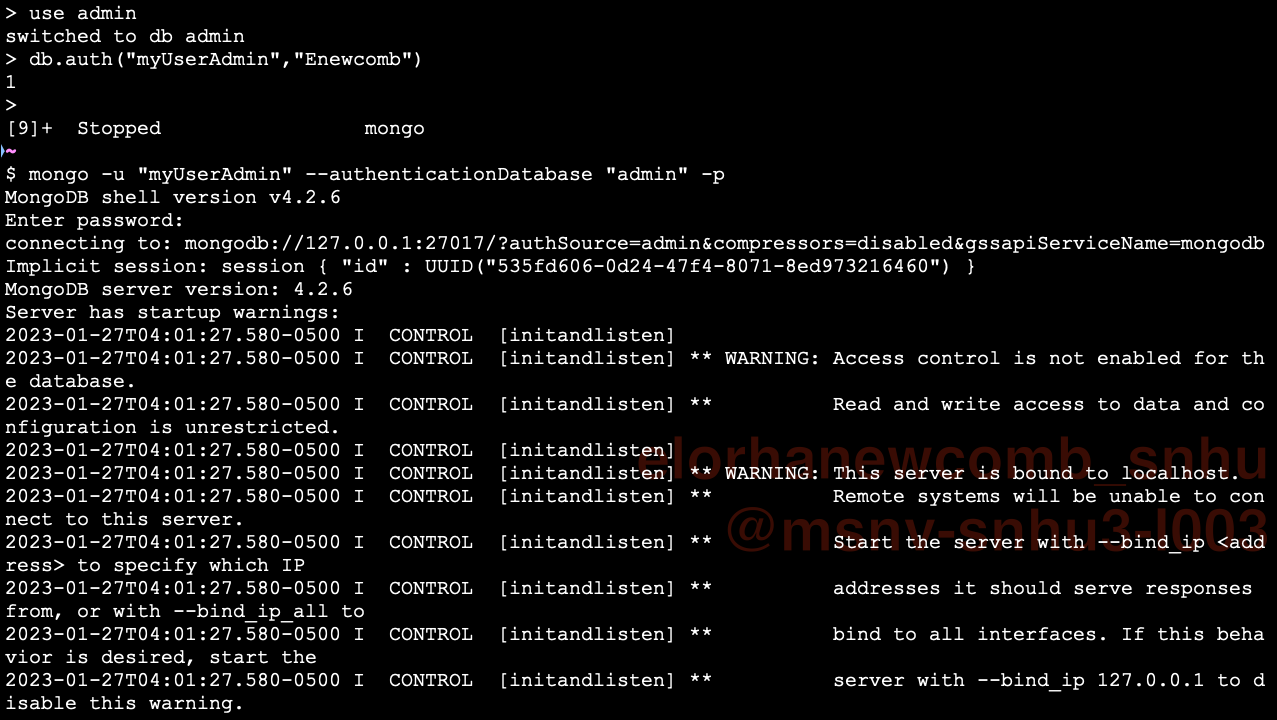


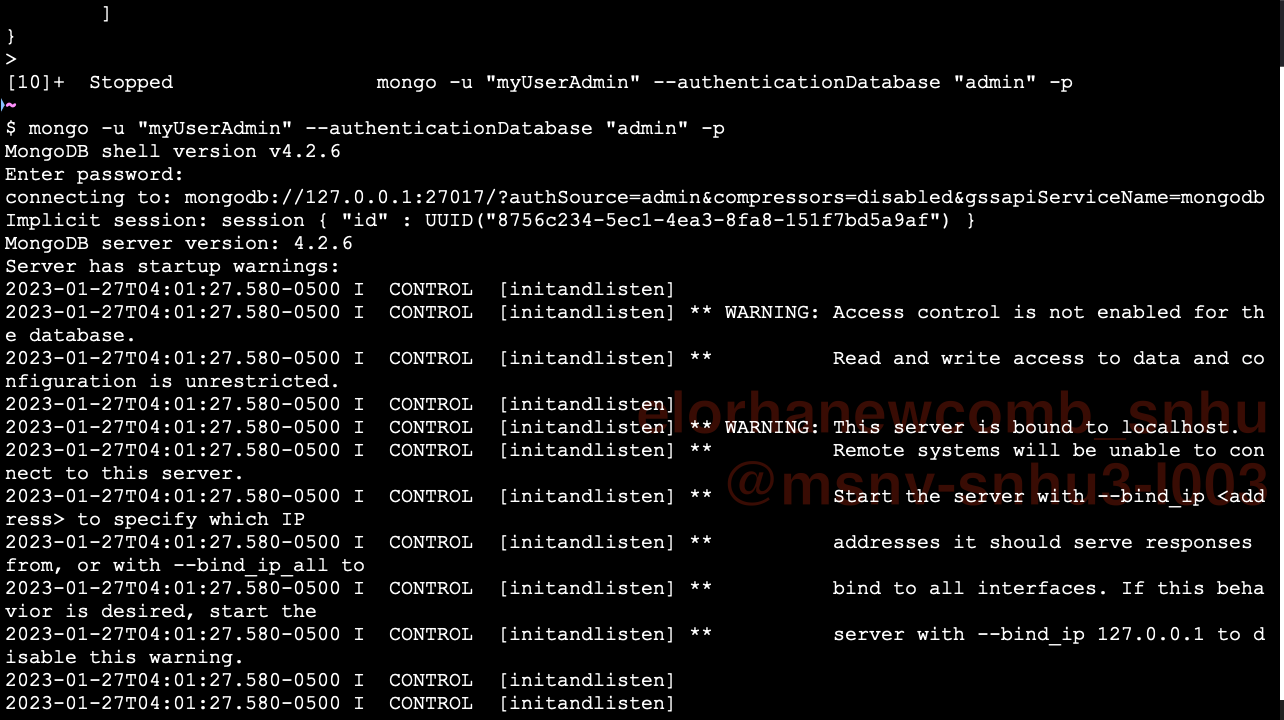
1. Create an Admin and a User accounts:



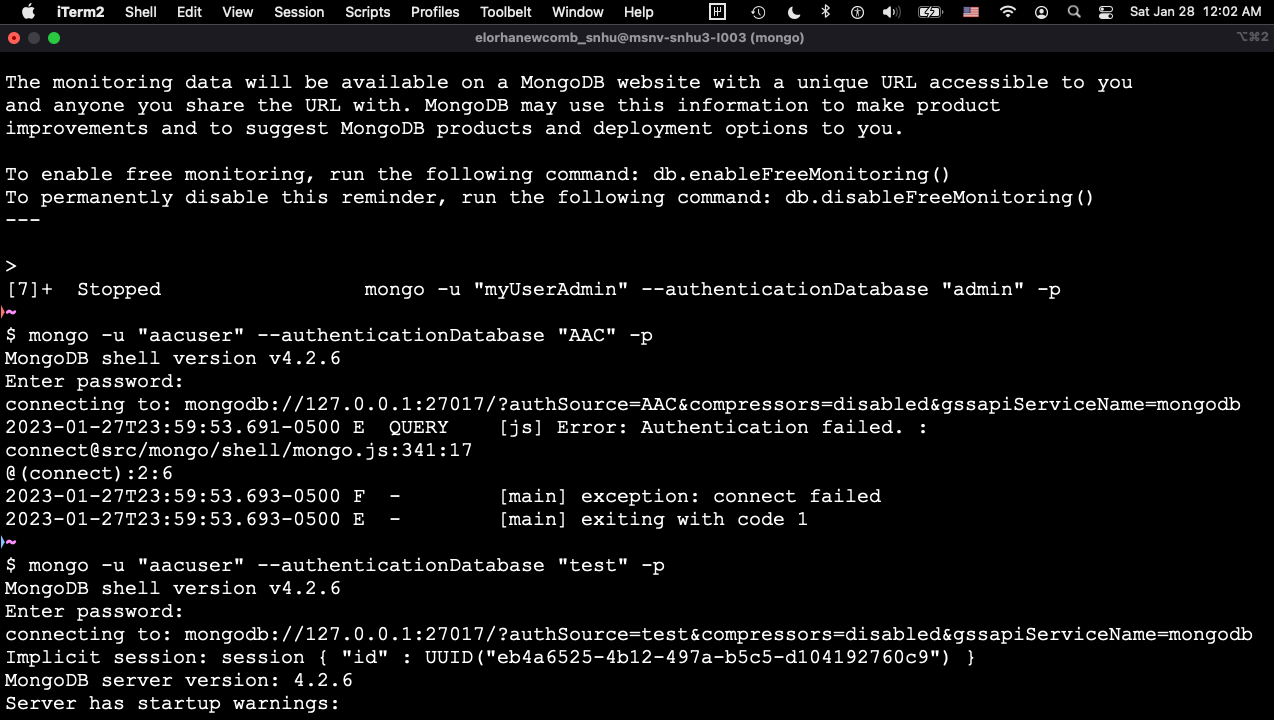












1. Full CRUD Implementation:

from pymongo import MongoClient

from bson.objectid import ObjectId

class AnimalShelter(object):

   """ CRUD operations for Animal collection in MongoDB """

   def \_\_init\_\_(self, username=None, password=None):

       # Initializing the MongoClient. This helps to

       # access the MongoDB databases and collections.

       # self.client = MongoClient('mongodb://localhost:27017')

       self.client = MongoClient('mongodb://%s:%s@localhost:29993/?authMechanism=DEFAULT&authSource=AAC'%  
(username, password))

       # where xxxx is your unique port number

       self.database = self.client['AAC']

# Complete this create method to implement the C in CRUD.

   def create(self, data):

       if data is not None:

           insert = self.database.animals.insert(data)

           print("Task Finished")# data should be dictionary

           if insert !=0:

               return True

       else:

           raise Exception("Nothing to save, because data parameter is empty")

# Create method to implement the R in CRUD.

   def read(self, data):

       if data is not None:

           data = self.database.animals.find(data, {'\_id':False})

           return data

       else:

           raise Exception("Nothing to search, data parameter is empty")

# Create method to implement the U in CRUD.

   def update(self, data, param):

       if data is not None and type(data) is dict:

           if param is not None and type(param) is dict:

               # If data and param are dict, then update

               self.database.animals.update\_one(data, param)

               # Store update in variable

               animal = self.database.animals.find\_one(data)

           else:

               raise Exception("Nothing to save, parameter is empty")

       else:

           raise Exception("Nothing to save, parameter not type dict")

       return animal

# Create method to implement the D in CRUD.

   def delete(self, data):

       if data is not None and type(data) is dict: # data should be dictionary

           print("\nDeleting", data)

           print(self.database.animals.delete\_one(data))

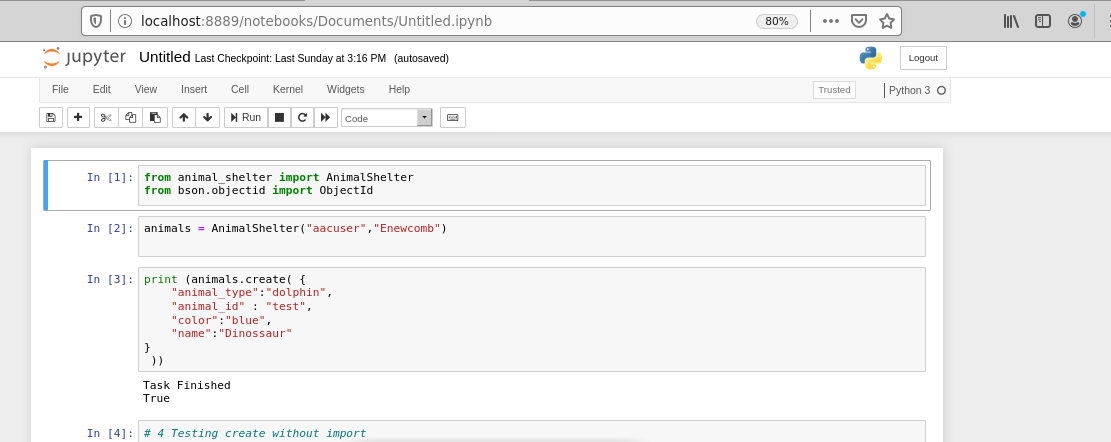
           print("\nEntry deleted", data)

       else:

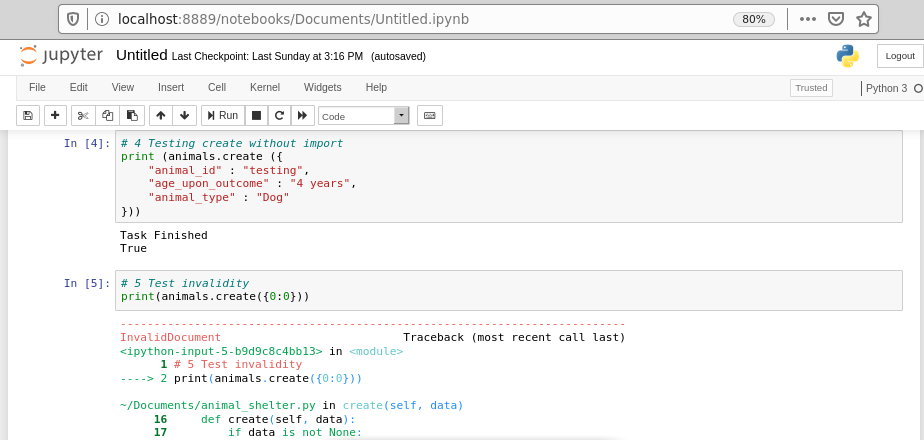
           raise Exception("Unable to delete entry")

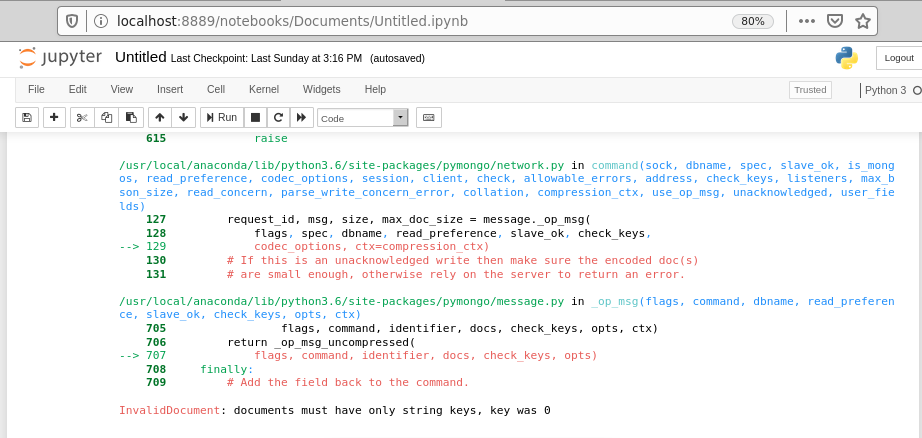
1. Script Testing:

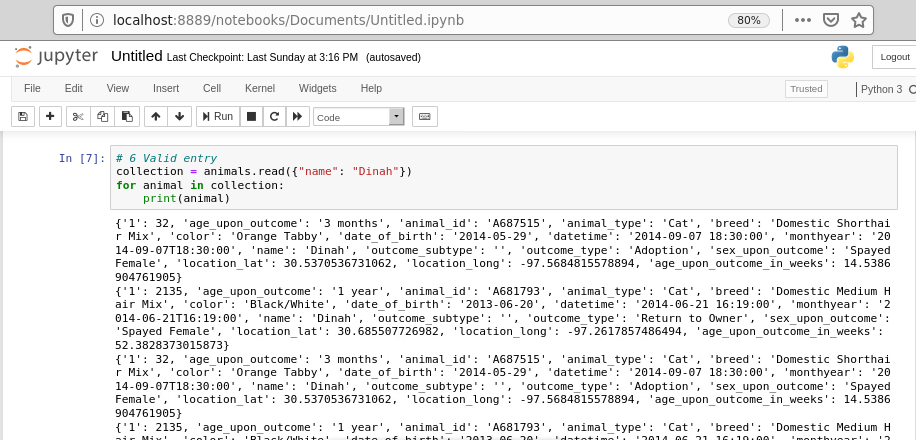
Testing Import



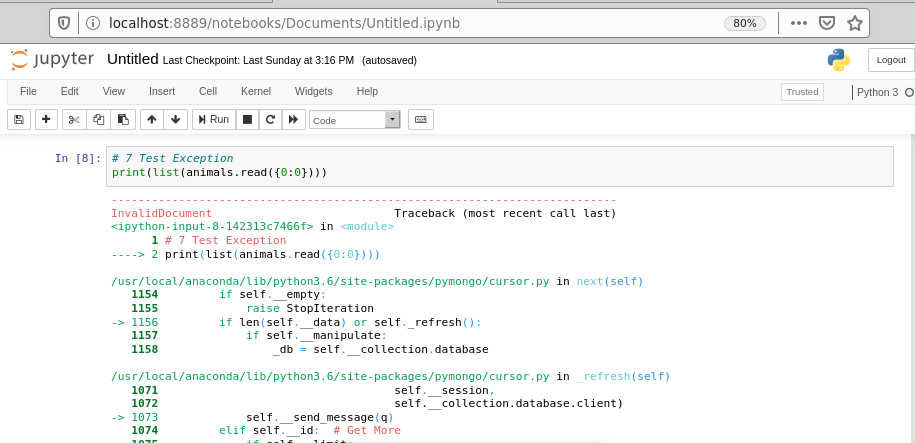
Testing Invalid Entry

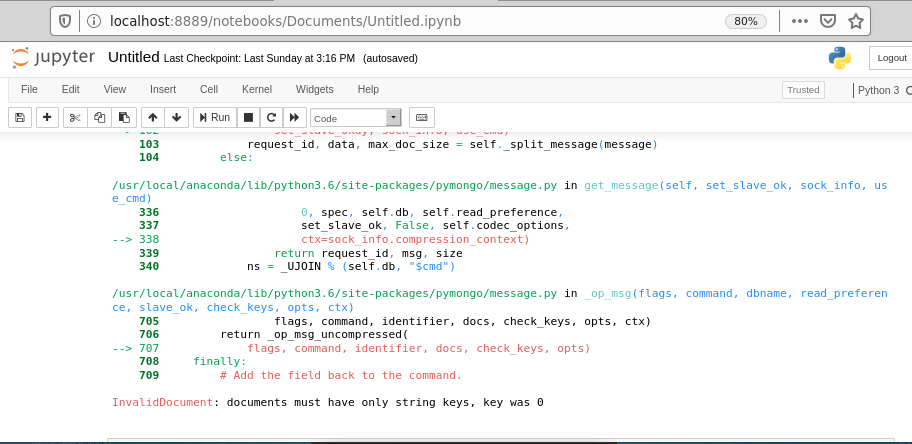




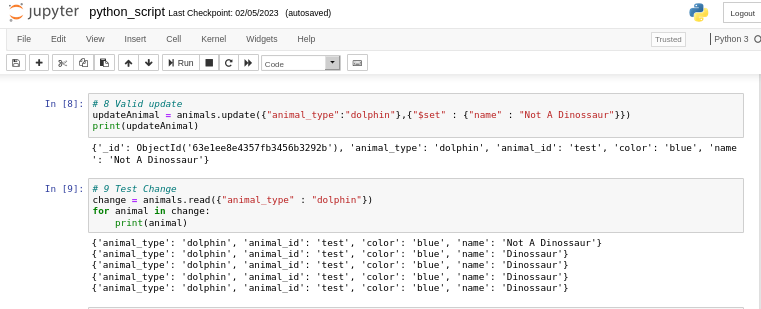
Testing Valid Entry  


Testing Exception





Testing Update



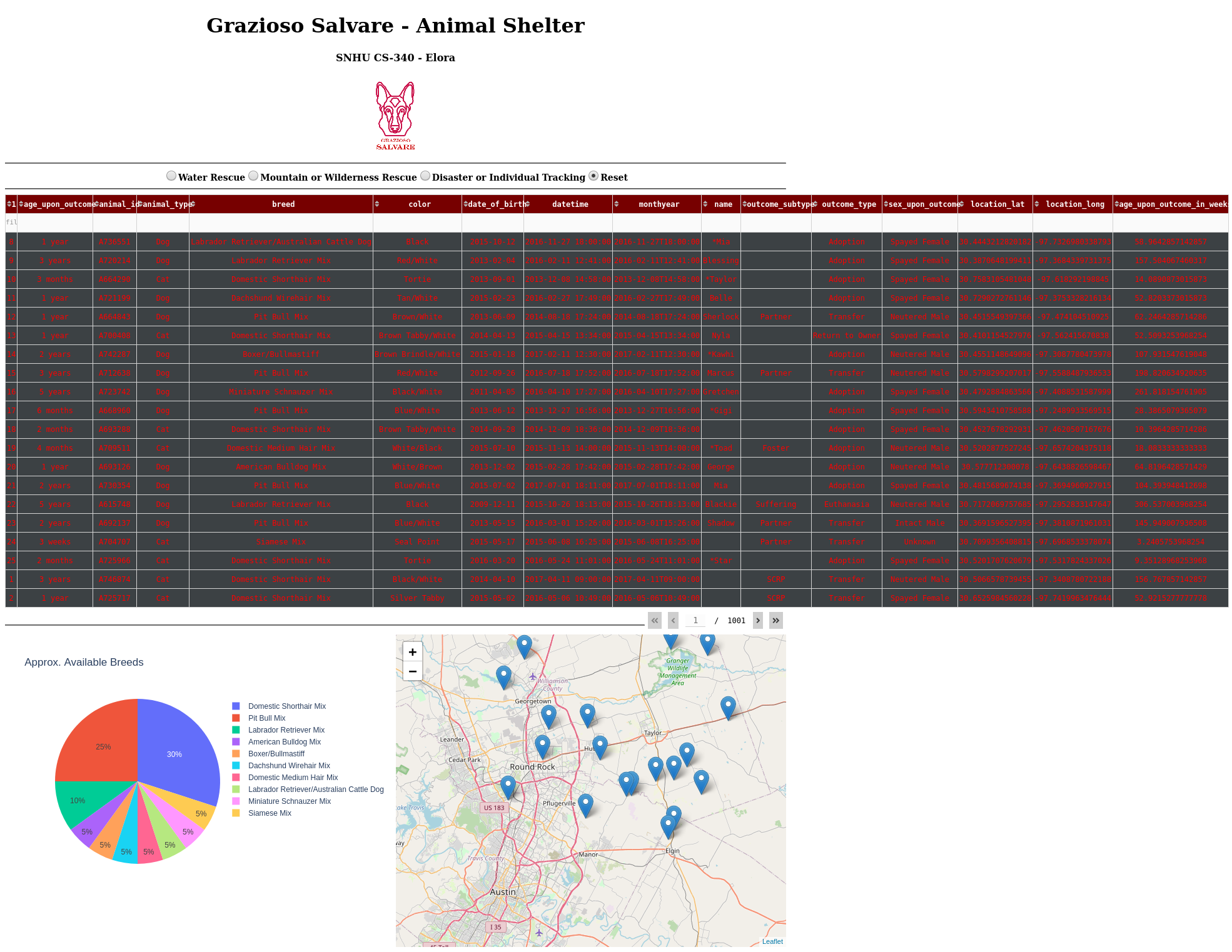
Testing Drop (Delete)

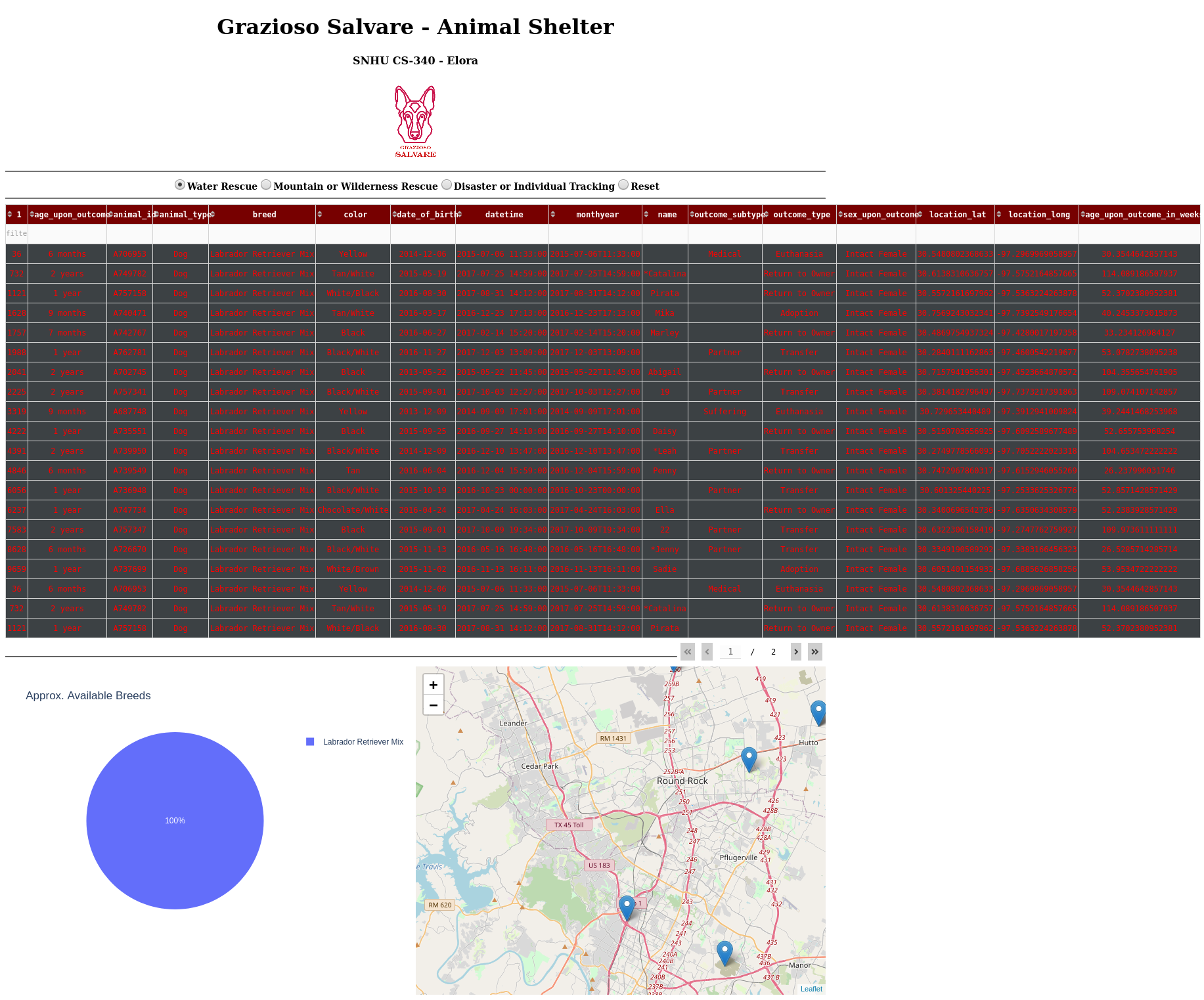


***[Web Application](/ProjectTwoDashboard.ipynb)***

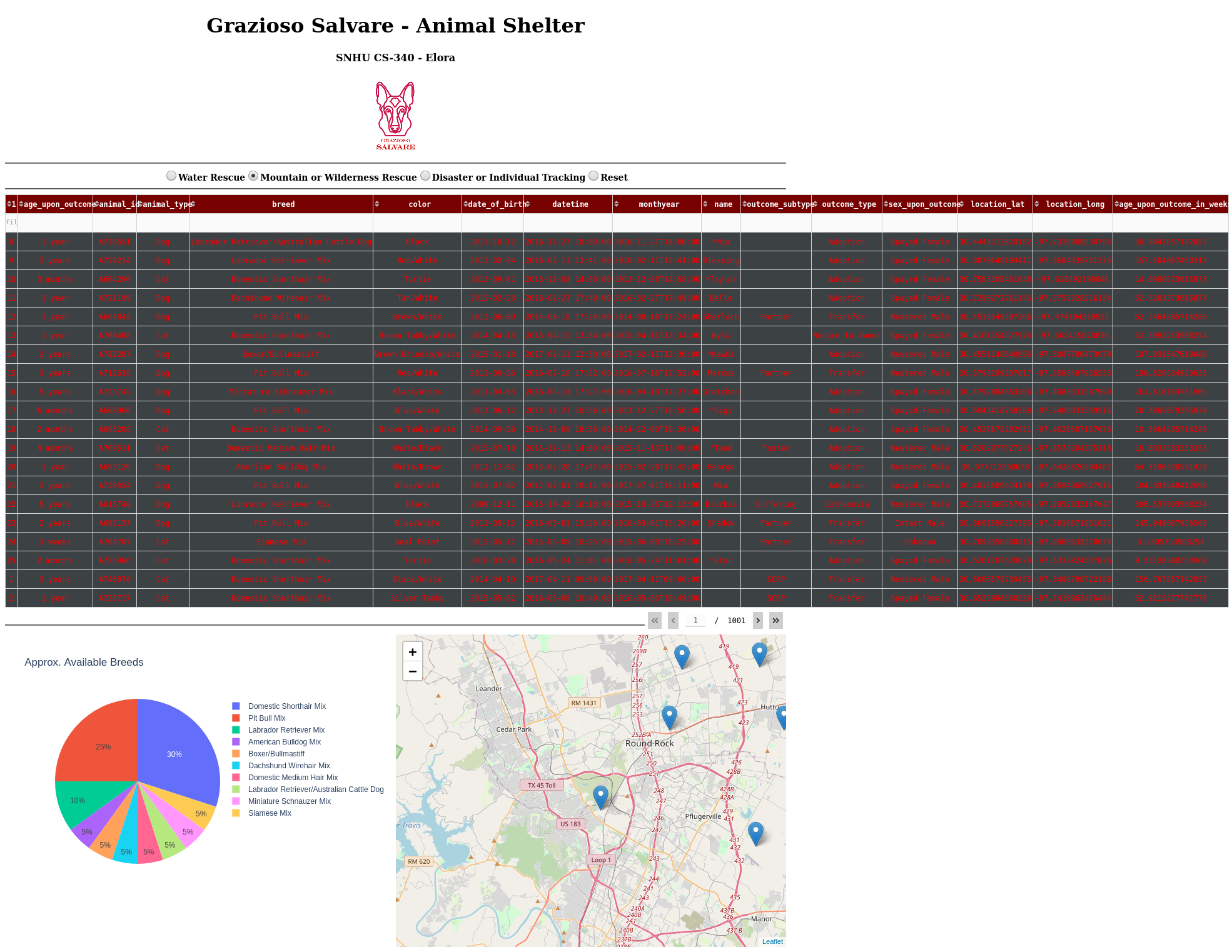
*The primary use case for this system is a browser-based dashboard that will display database data, as well as query options for finding animals for each specific type of service (water rescue, wilderness rescue, and tracking). In addition to implementing the provided Python module to utilize the read function, the service filters are supplied by the following MongoDB queries:*

***Filter Data Table by Header***

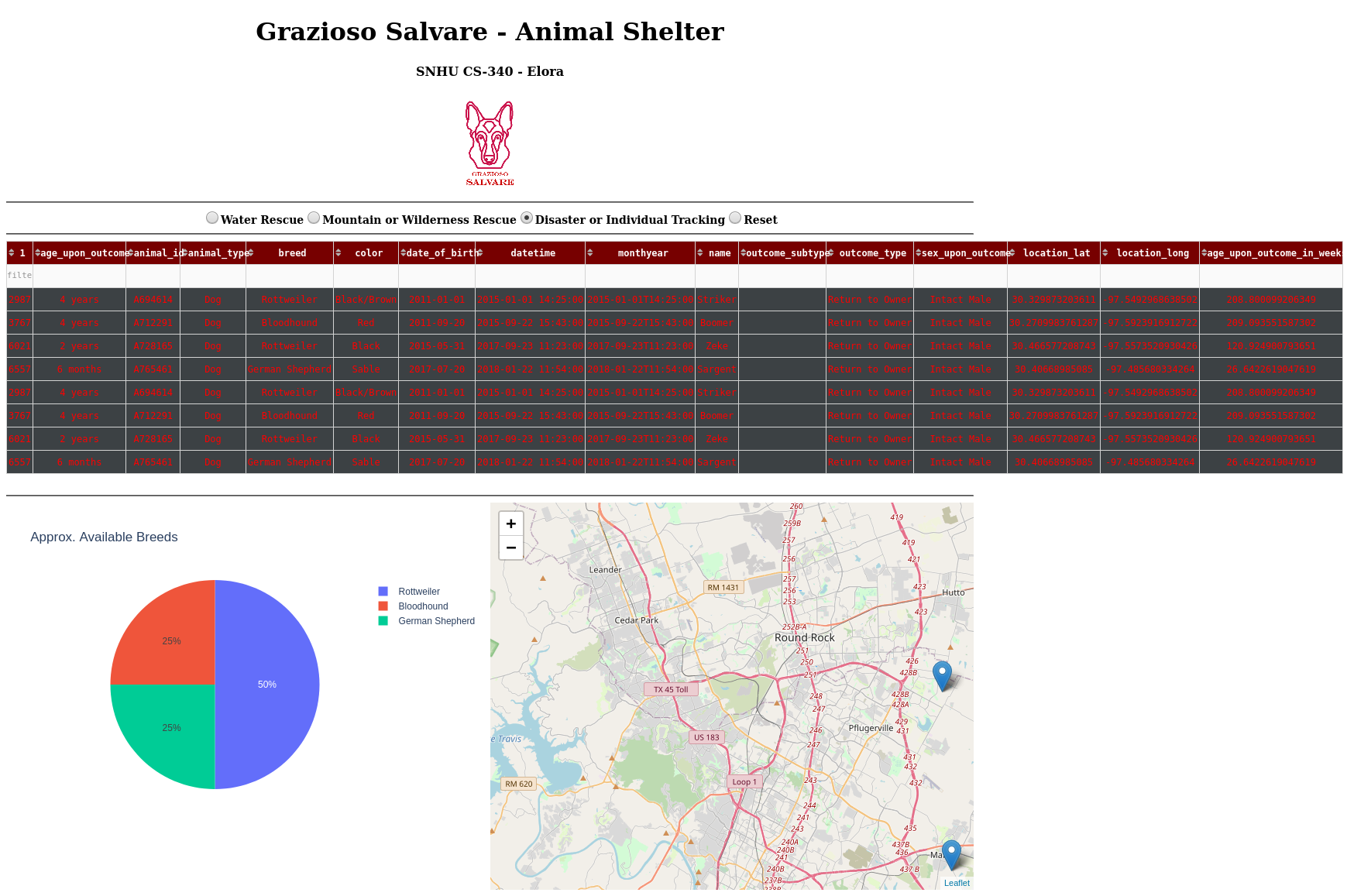
******

***Water Rescue***

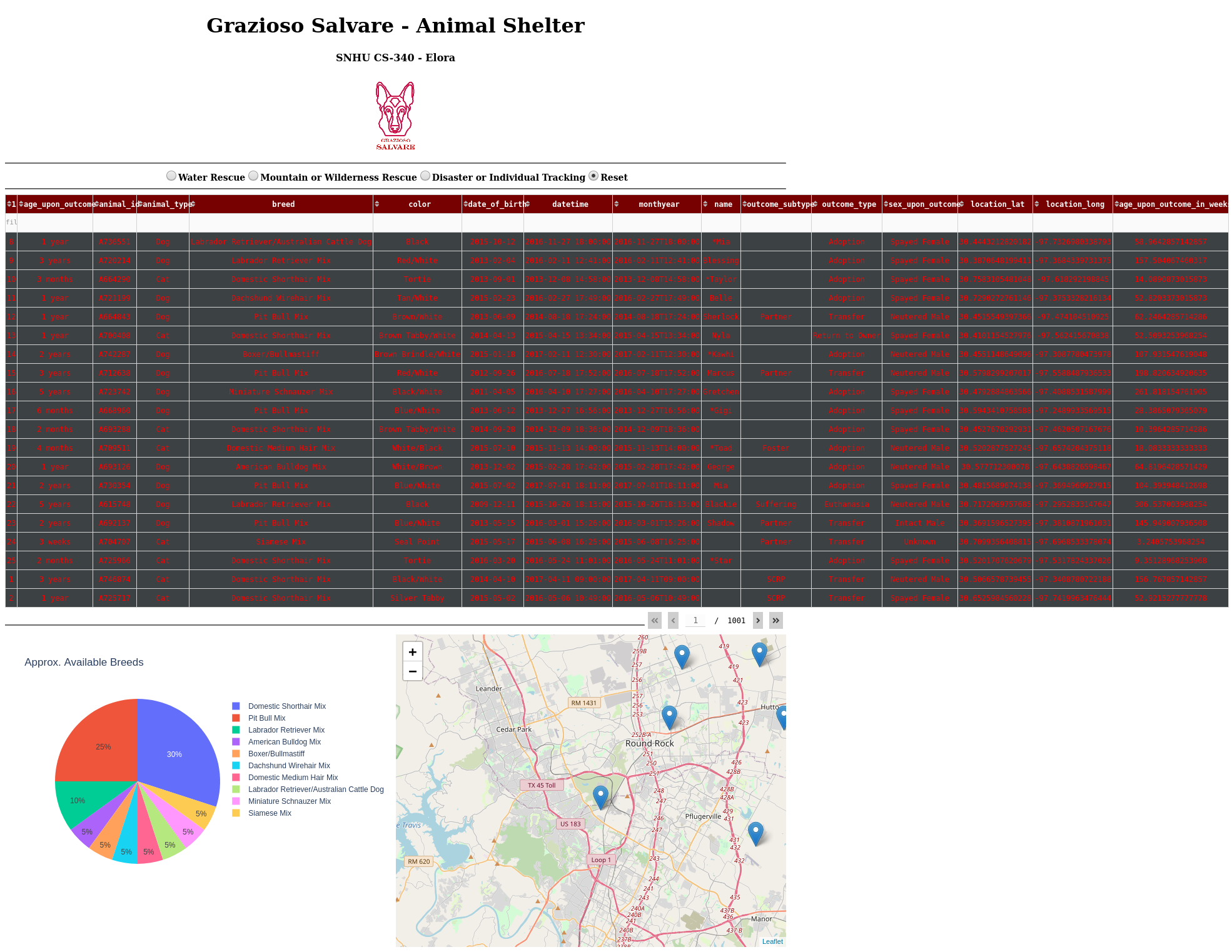
***Mountain or Wilderness Rescue Water Rescue Filter***

******

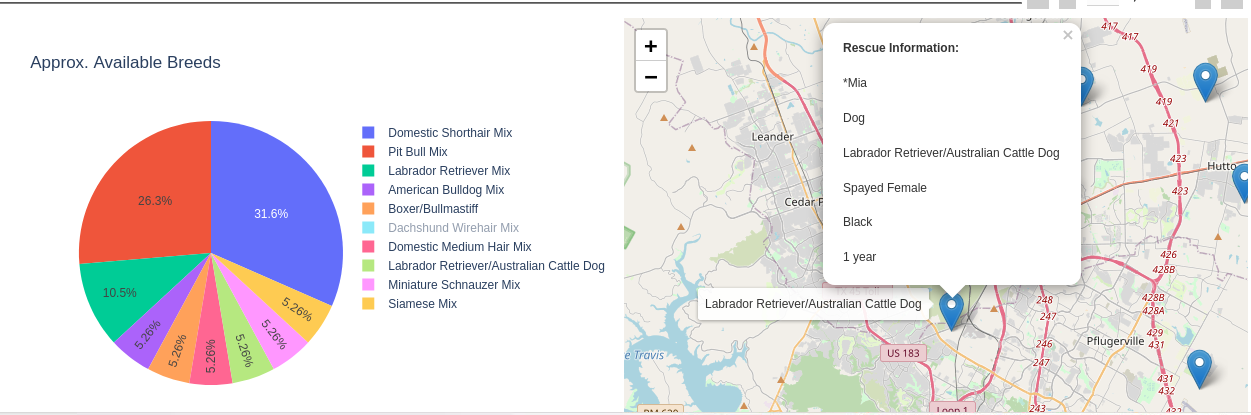
***Disaster Rescue and Individual Tracking***

******

***Reset All Filters***



***Map and Tooltip***



## Contact

Your name: Elorha Newcomb