# AAC Animal Database README

## About the Project

This project is designed to assist users with searching for good animal candidates to be trained in search and rescue. The software utilizes existing data from animal shelters in the Austin, Texas area and allows users to create, read, update, and delete information from the database. In this manner trainers can search for viable candidates for their search and rescue training, and the animal shelters can keep the list up to date by adding, modifying, and removing listings.

## Motivation

Search-and-rescue is a vital community service, and not all animals are suited to the task. It has been noted that certain breeds are more effective in certain environments, and dogs of a certain age and temperament are easier to train to be more effective. By utilizing data from shelters within the area, dogs (and other animals) may be identified to fit the profile the search-and-rescue team is looking for. This allows the trainers to have the best pool of candidates possible for effective search-and-rescue operations and gives these animals a new home and purpose.

## Getting Started

This project utilizes MongoDB for the database and python for the application. The dataset was supplied in the format of a csv file. Jupyter Notebook was utilized to create the front end of the application with which the user interacts with.

Launch Mongo using the Terminal application in Linux.

Upload the dataset into the database. For our purposes the dataset was uploaded into the ‘AAC’ database under ‘animals’.

Create an administrator account for the database. This will be used to create user accounts.

Create user accounts for the database as appropriate, and ensure they have the appropriate permissions. Animal shelters need to be able to read and write, where trainers should be able to read, but not make any changes to the database.

Exit mongo and relaunch with authentication. This will ensure that only users with the appropriate authentication can make modifications to the data.

A python code was created to create, read, update, and delete documents from the database. The create function takes ‘data’ in the form of a dictionary. Where the read function takes query information, as when using the find() function in Mongo. See the Usage section for examples of this code. The update function uses the find\_and\_modify() function to update part or all of the first document that matches the find criteria, and returns the updated document. Finally the delete function uses the find\_one\_and\_delete() function to find the first document that matches the criteria and deletes it, returning the document that was deleted.

Jypyter Notebook was used to create the front end application, utilizing Dash for an interactive table and display. This application should make use of the middleware, the python code described above. The data is imported into an easily readable table with options to filter the data using preset options. These options are accessed through a drop down menu above the table. This will also update a pie graph displaying the percent of dog breeds that align with the criteria. Furthermore, selecting a particular dog on the left of the screen will bring up a map with the geolocation of the animal.

## Installation

-MongoDB

-MongoClient

-MongoImport

-PyMongo

-Jupyter Notebook

## Usage

### Code Example

# Create Method

def create(self, data):

if data is not None:

create = self.database.animals.insert(data) # data should be dictionary

if create == 0:

return False

else:

return True

else:

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

# Read Method

def read(self, query):

if query is not None:

if query:

queryResult = self.database.animals.find(query)

return queryResult

else:

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

This code initializes an instance of the MongoClient, accepting a username and password to the method, and directs to the appropriate database. The create function takes in data, typically as a dictionary. The read function takes in a query using key/value pairs. The update function takes in a query of key/value pairs and the information to be updated in key/value pairs. The delete function takes in a query of key/value pairs.

### Tests

Each Method should be tested independently. First you want to set up your testing data. For our purposes, we will be setting up an animal entry to be created and a query to be searched. This will require defining an animal with all of the appropriate dictionary entries, and a query using a key/value pair, and a key/value pair to update. The read method will return all applicable entries, but we will restrict our testing to only look for the entry we created. This will help verify that it was created properly. The update method will be tested by using the same query information as the read function and updating part of the data in the document that was created. This method should return the document as a readable json format, so that the updated information can be verified. Finally, the delete method will use the same query information and delete the document that has been created and modified. This will return the deleted document as a readable json format, so that it can be verified the appropriate file was removed.

After everything has been defined, run the first test to initialize the database. This will use the AnimalShelter method and take a string for the username and password. If the user does not the appropriate credentials, the following tests will fail.

Execute the create method using the animal entry that was defined initially. This will return True if it was created successfully. Next, execute the read function using the query that was defined initially. This will return a cursor object if it was successful. Then execute the update function, which will return the readable information. Verify that this was updated appropriately. Finally execute the delete function, which also returns the readable information of the document that was deleted.

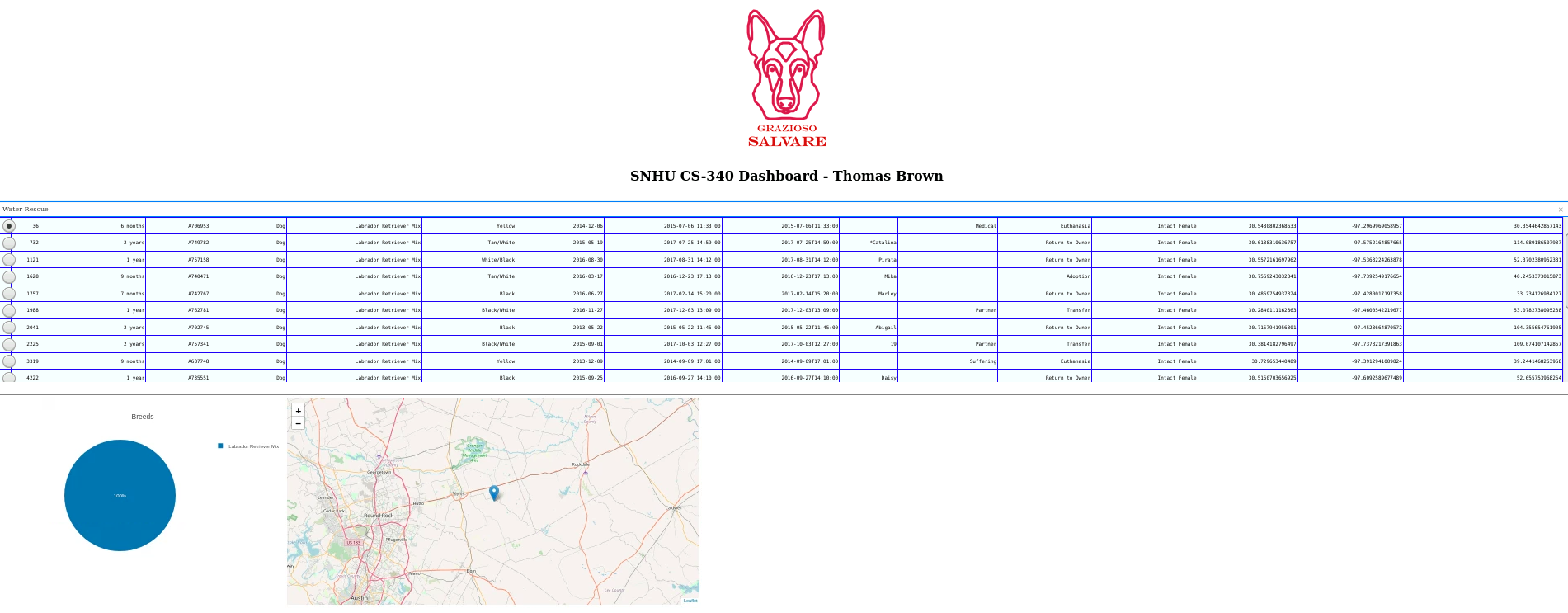
### Testing should also be performed on the final application. It is recommended to start with the display of the table, then add widgets one at a time to ensure everything remains functional.

### Screenshots

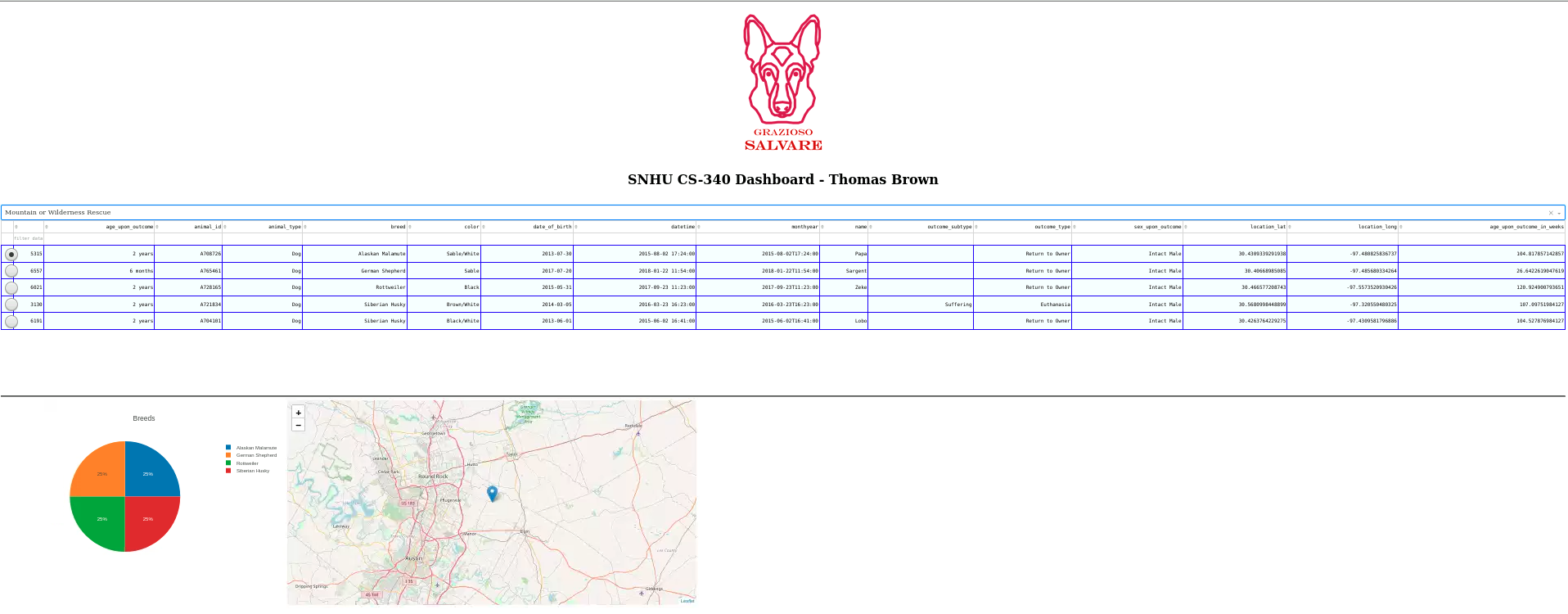
Screenshot #1: Dashboard Initial State (Zoomed Out to Display Entire Page)



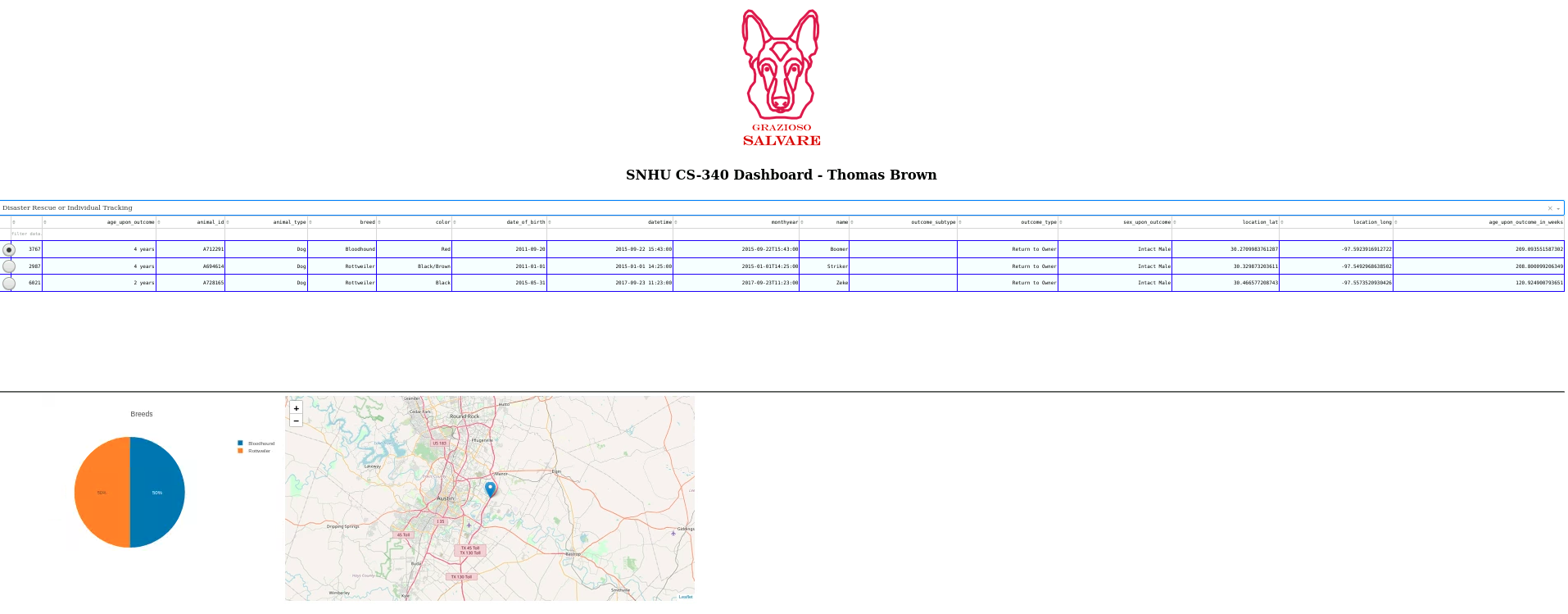
Screenshot #2: Water Rescue Filter



Screenshot #3: Mountain or Wilderness Rescue Filter



Screenshot #4: Disaster Rescue or Individual Tracking Filter



Screenshot #5: No Filter / Reset Option

## 

## Contact

Thomas Brown