# CS 340 README Template

*Prerequisite: Ensure jupyter is installed and configured to your environment.*

## About the Project/Project Title

Python CRUD Mongodb app implements CRUD operations for an Animal collection in MongoDB. CRUD operations currently implemented are Create, and Read. Still to be implemented are Update, Delete.

## Motivation

Python allows for secure MongoDB user authentication and connection to a MongoDB client instance, database selection, user and password passing upon instantiation, and searching through the database and inserting entries to the database through Python dictionaries.

**Rationale for MongoDB**

The rationale for choosing MongoDB as the model component database for Python include it being schema-less (no strict fields), single object structure clarity, no complex joins as in SQL, dynamic querying, an ability to index any attribute and format it to a table.

**Contents of Readme:**

**PART 1: Creating DB, authenticating users, testing mongodb queries (Mongodb database is the MODEL in MVC)**

**PART 2: Implement Dash shell, test front-end queries with interactive filter options using implemented Pymongo CRUD Module and DASH. (Our pymongo CONTROLLER uses Dash widgets for an interactive VIEW)**

**PART 3: Implement dashboard widgets that receive input from the interactive options and present those dynamic updates to the client, including a geolocation chart and a Pie chart)**

## PART 1: Creating DB, authenticating users, testing mongodb queries (Mongodb database is the MODEL in MVC)

To get a local copy set up, follow these example steps:

1. Download main.py and AnimalShelter.py files, these will be used to interact the with mongoDB instance.
2. Ensure you have a working MongoDB database collection imported, with user Authentication enabled for user “aacuser”, given readWrite priveleges over the “AAC” database. Note the address and port MongoDB is connecting to. For example, 'mongodb:// 127.0.0.1:49727/AAC'
3. Ensure correct “user” and “password” are initialized in the animal\_shelter object as these will be passed to MongoDB client.
4. Ensure correct database is set in self.database = self.client["AAC"] line.
5. Create an entry by passing a data entry (dictionary) to animal\_shelter.create(data)
6. Read an entry by passing a query (dictionary) to animal\_shelter.read(query)

## Installation

1. *Python 3.6.9 :: Anaconda, Inc. Python will be used within both the CRUD module and the Dash interface, acting as the Controller and View*
2. *Pymongo (pip package 3.10.1)*

* *pip install pymongo*
* *pip install jupyter\_plotly\_dash==0.4.2*
* *pip install django-plotly-dash==1.3.1*
* *pip install dash==1.10.0*
* *pip install dash-leaflet==0.0.9*

***TODO: Rationale behind each tool***

*Pymongo is to be used to allow pythonic interaction with out mongo database, jupyter\_plotly\_dash and django-plotly-dash allows user interface functionality as well as a Jupyter notebook environment, and dash-leaflet supports map display functionality.*

1. *MongoDB v4.2.6*

MongoDB is to be used as the Model in our MVC, holding the animal data securely. Ensure appropriate Python and MongoDB packages are installed for appropriate operating system and ensure Python is in the System Path so we can run the files.

## Usage

### Code Example

*create(data):* function implementes Create in Crud, passing a python dictionary of MongoDB data will use insert\_one to create a new db entry.

*read(search\_crit=None):* If passed a search criteria in form of a dictionary, will search based on that MongoDB query information; if passed nothing, will read all entries.

### Tests

***Example usage:***

*animal\_db\_entry = [*

*{"name": "Felix", "type": "cat"},*

*{"name": "Garfield", "type": "cat"},*

*{"name": "Tom", "type": "cat"}*

*]*

*for animal in animal\_db\_entry:*

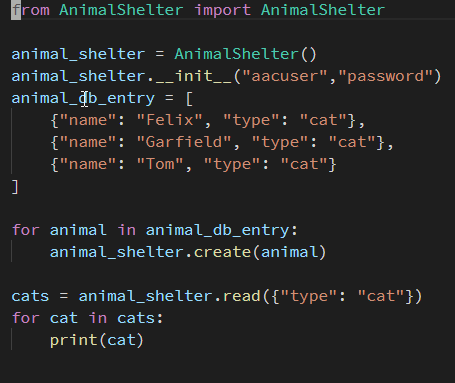
*animal\_shelter.****create****(animal)*

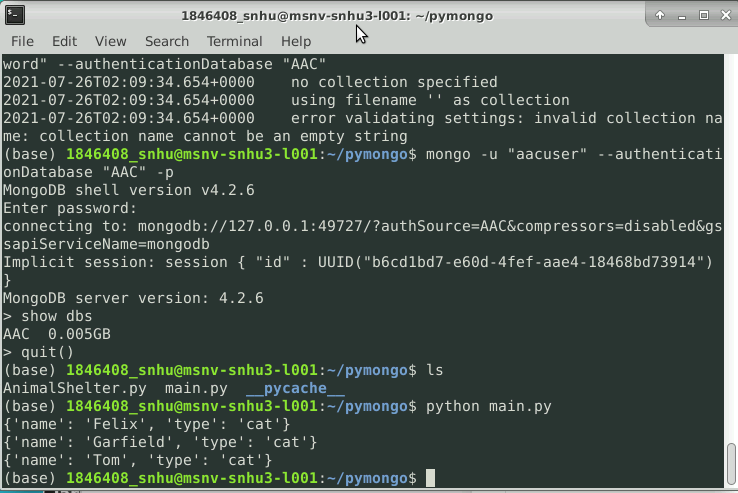
*cats = animal\_shelter.****read****({"type": "cat"})*

*for cat in cats:*

*print(cat)*

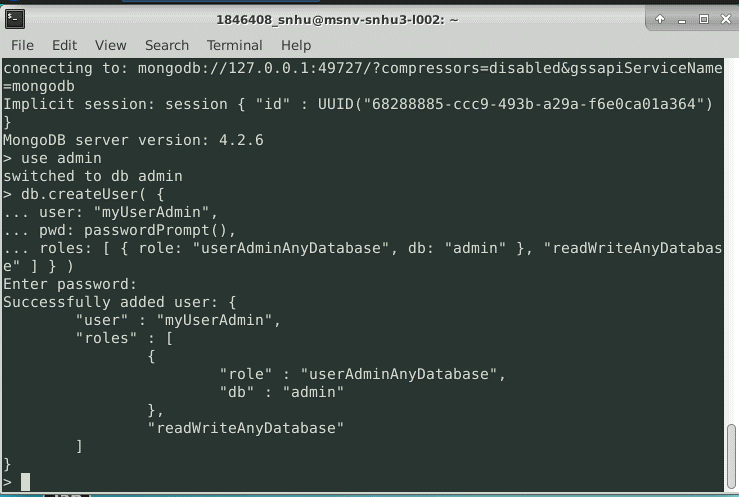
### Screenshots





1. Creating an administrator and user account in the mongo shell to ensure user authentication and collection was created.

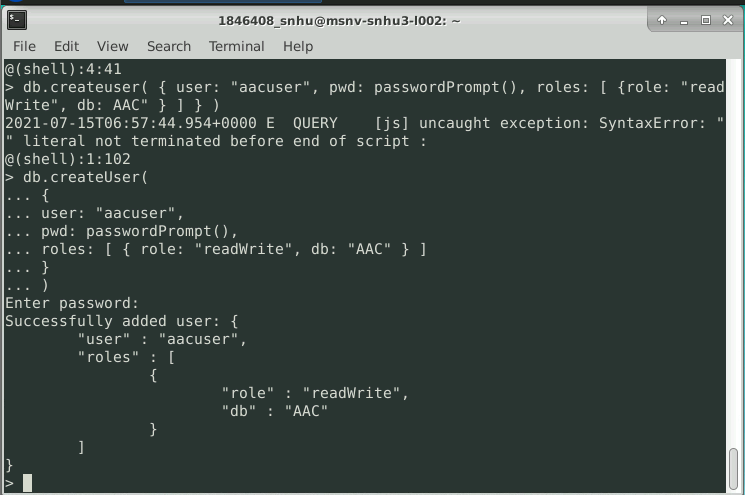
First, ensure to start the server with the **noauth** flag and run mongo:



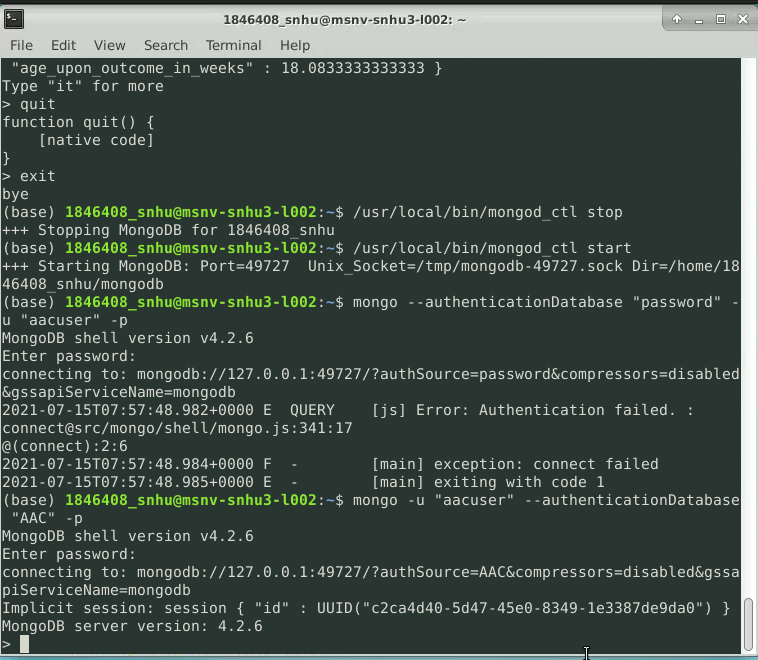
Example usage of logging in as admin:



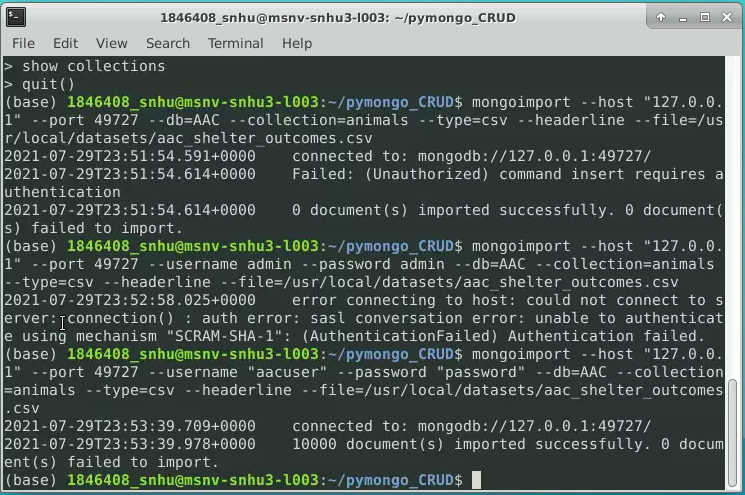
Creation of aacuser account:

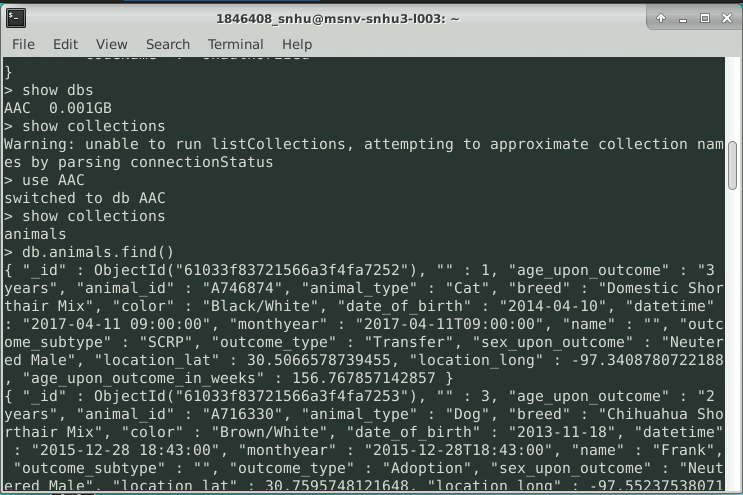


Example of logging in as aacuser:

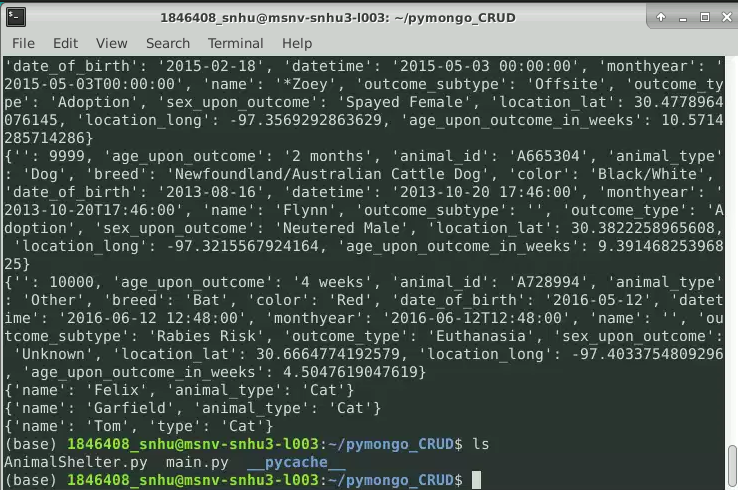


1. Inserting a CSV using mongoimport, including proper host url, port number, username, and password:

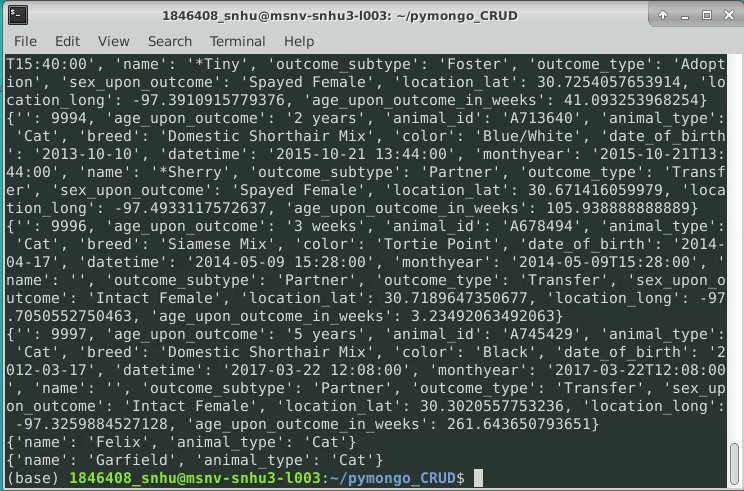


Next, ensure the AAC database is properly created and imported from the csv file of values

Example of a Read operation: all\_animals = animal\_shelter.read({})

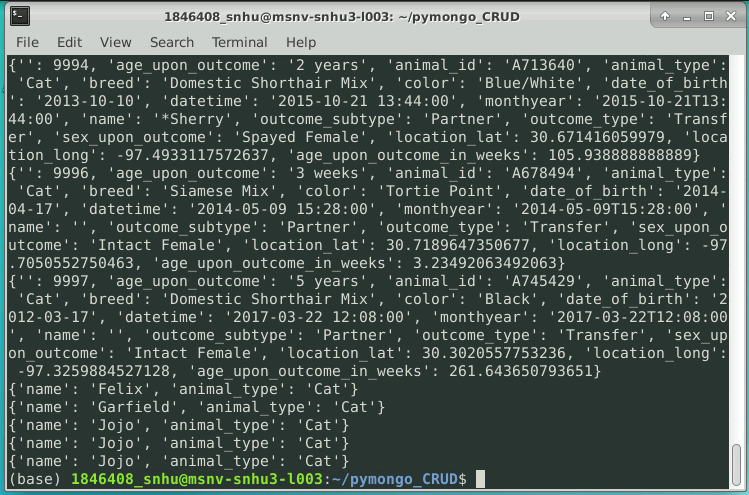


After a Delete operation is performed: animal\_shelter.delete(“name”: “Tom”)



Note Tom is removed from the list.

Next, perform example of the update functionality: update\_cats = animal\_shelter.update(animal\_db\_entry2) Note since \_id is not given, alternative functionality of adding entry is performed, duplicates allowed.



**PART 2: Implement Dash shell, test front-end queries with interactive filter options using implemented Pymongo CRUD Module and DASH. (Our pymongo CONTROLLER uses Dash widgets for an interactive VIEW)**

Ensure the Python CRUD module is imported into the Dash IPYNB file:

from AnimalShelter import AnimalShelter

username = "aacuser"

password = "password"

shelter = AnimalShelter(username, password)

The following queries will be used in the pymongo Dash interface in responding to user input:

**QUERY FOR ‘WATER’**

db.animals.find(

{

"breed": {

"$in": [

"Chesapeake Bay Retriever",

"Labrador Retriever Mix",

"Newfoundland"

]

},

"sex\_upon\_outcome": "Intact Female",

"age\_upon\_outcome\_in\_weeks": {

"$gte": 26.0

},

"$and": [

{

"age\_upon\_outcome\_in\_weeks": {

"$lte": 156.0

}

}

]

}

)

**QUERY FOR ‘Mountain or Wilderness’**

db.animals.find(

{

"breed": {

"$in": [

"German Shepherd",

"Alaskan Malamute",

"Old English Sheepdog",

"Siberian Husky",

"Rottweiler"

]

},

"sex\_upon\_outcome": "Intact Male",

"age\_upon\_outcome\_in\_weeks": {

"$gte": 26.0

},

"$and": [

{

"age\_upon\_outcome\_in\_weeks": {

"$lte": 156.0

}

}

]

}

)

**QUERY FOR Disaster or Individual Tracking**

db.animals.find(

{

"breed": {

"$in": [

"Doberman Pinscher",

"German Shepherd",

"Golden Retriever",

"Bloodhound",

"Rottweiler"

]

},

"sex\_upon\_outcome": "Intact Male",

"age\_upon\_outcome\_in\_weeks": {

"$gte": 20.0

},

"$and": [

{

"age\_upon\_outcome\_in\_weeks": {

"$lte": 300.0

}

}

]

}

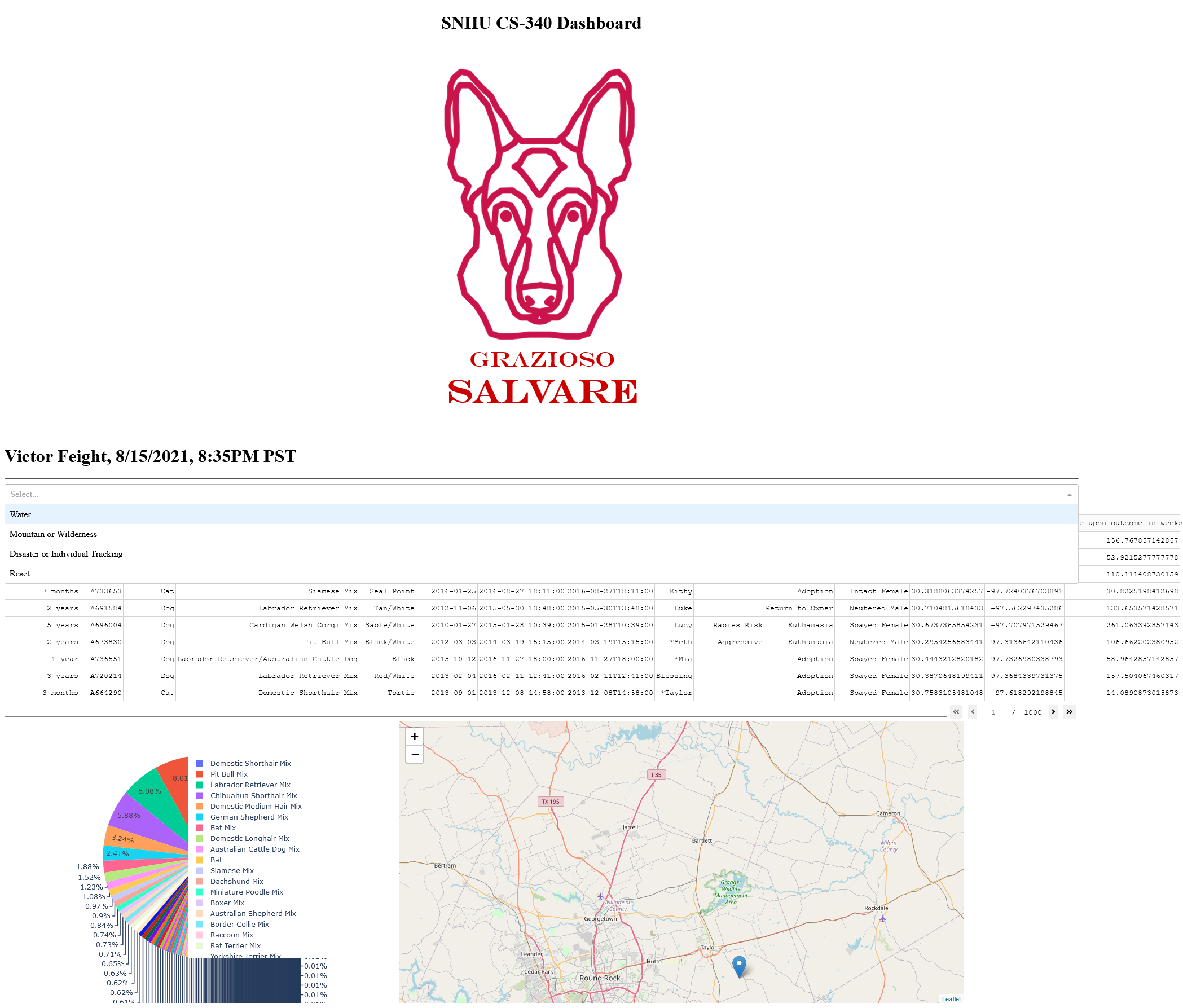
)

These queries are passed to the MongoDB via callback functionality within the Dash view.

**PART 3: Implement dashboard widgets that receive input from the interactive options and present those dynamic updates to the client, including a geolocation chart and a Pie chart)**

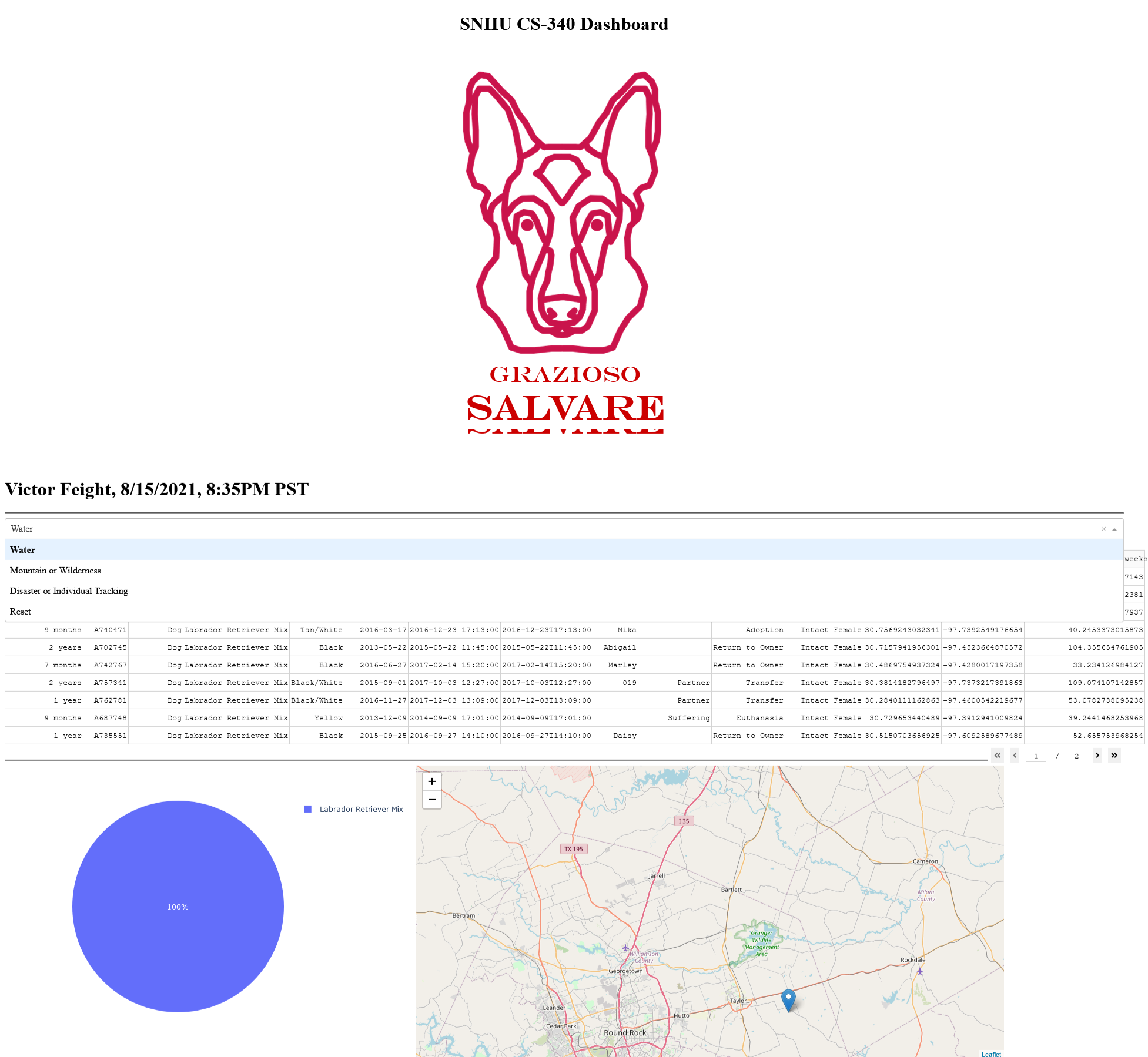
As proof of required functionality achieved, the following are screenshots showing the starting state following by execution of the dashboard components, including my piechart and geomap which responds to input via a dropdown menu containing the pymongo queries.

The starting state of your dashboard

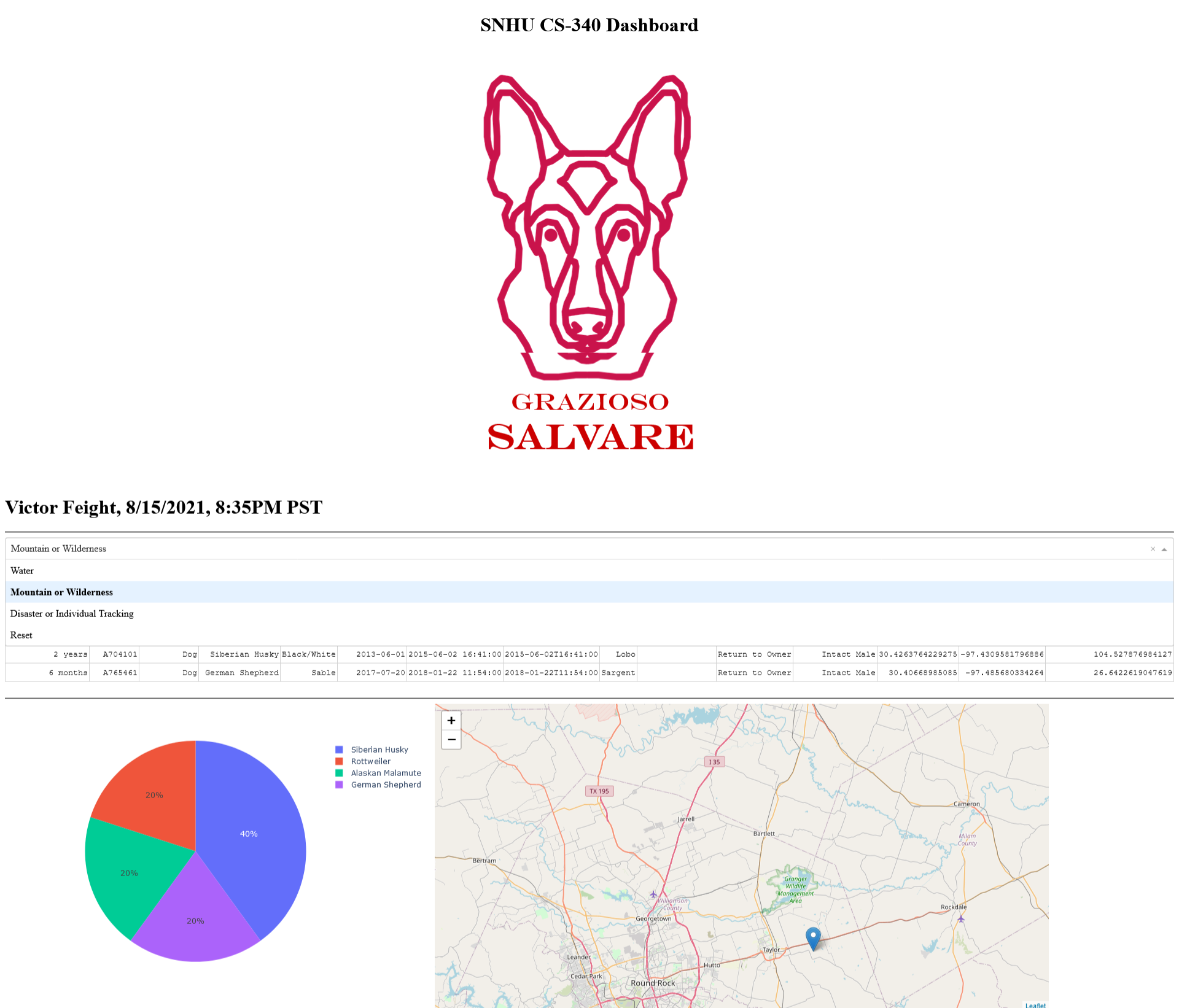


Executions of your dashboard (with mongodb filters applied by Dash dropdown widget)

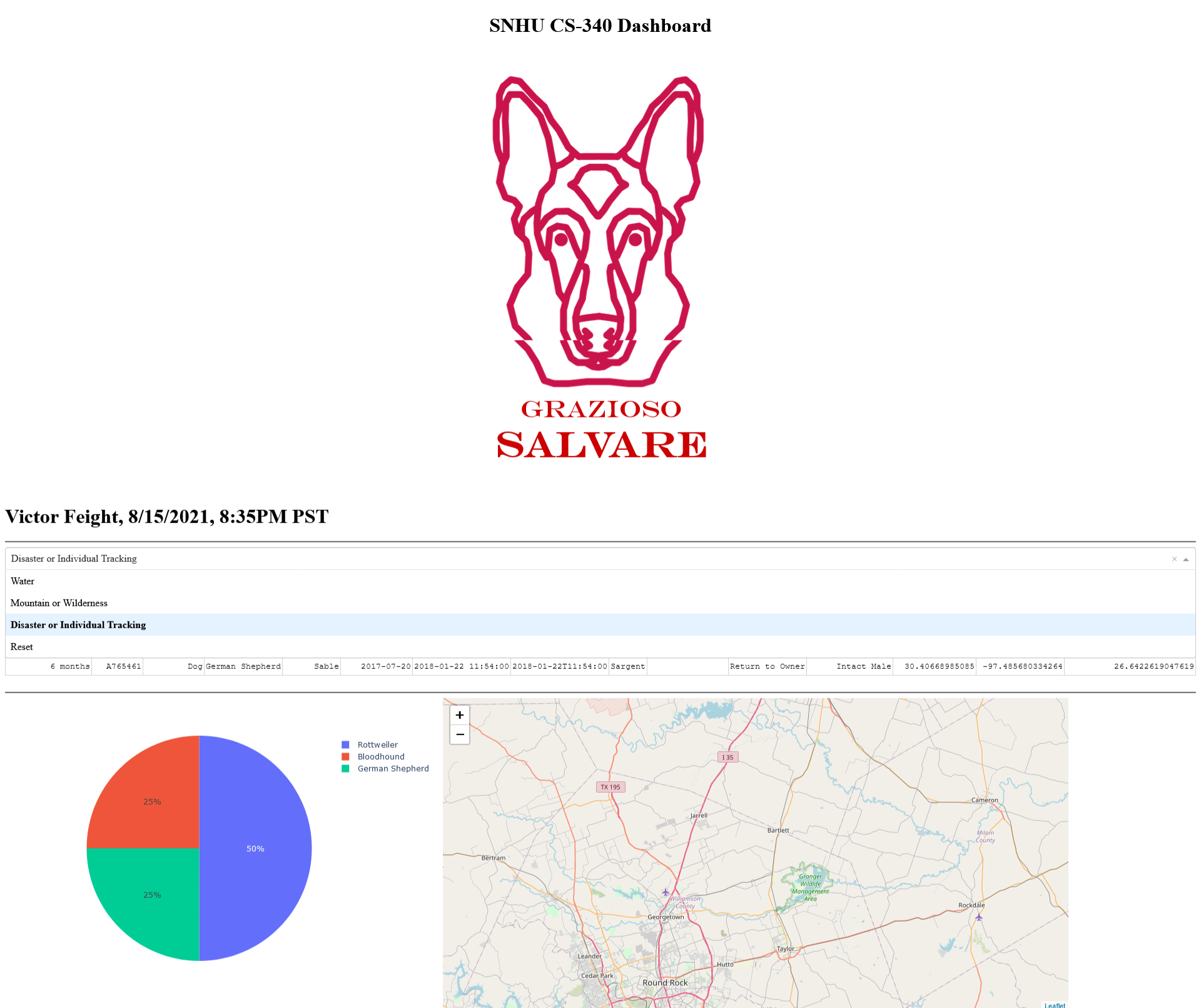
Water Rescue



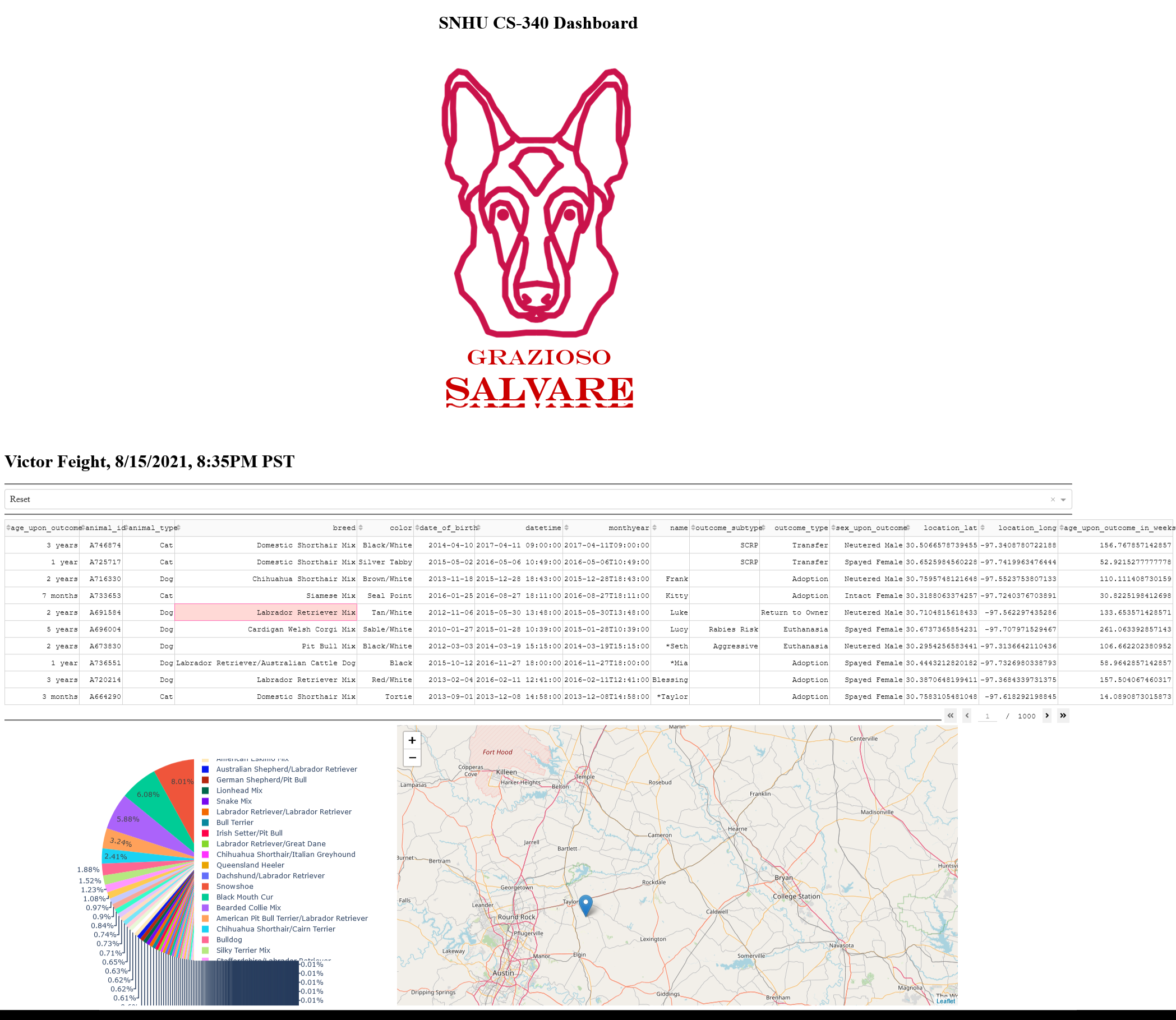
Mountain or Wilderness Rescue



Disaster or Individual Tracking



Reset (returns all widgets to their original, unfiltered state)



## Roadmap/Features (Optional)

Features still planned: Aggragation functionality

## Contact

Your name: Victor Feight, victor.feight@snhu.edu