# Project Two: Grazioso Salvare Dashboard

## About the Project

This project uses a middleware Python script, MongoDB, and Dash to give users Create, Read, Update, and Delete functionality (CRUD) within a MongoDB database, and then displays the results on an interactable web application dashboard. Specifically, pulling data from the Austin Animal Center (AAC) database. The web application script uses pre-written queries to display the necessary data on the dashboard. The dashboard includes a data table, pie chart, and geolocation map to display filtered data from the AAC database.

## Motivation

The motivation for this project is to enable users to easily interact with the MongoDB database using functions within the CRUD library, and quickly and efficiently manipulate data. This then allows for accurate data to be displayed on a user-friendly dashboard according to Grazioso Salvare’s breed and rescue type specifications.

## Getting Started

To begin, ensure that MongoDB and Python are installed on your system, as well as necessary Python libraries.

Import the AAC outcomes data set to MongoDB using the file “aac\_shelter\_outcomes.csv”.

Finally, import the CRUD python file.

In MongoDB you must create a user which has the correct permissions to interact with the database. In testing, I created a user ‘aacuser’ which has read/write permission within the AAC database. The credentials for this user will then be used in the init function in the Python script to connect to MongoDB.

## Installation

Download the MongoDB installer for your platform at mongodb.com. MongoDB is a NoSQL database which is useful for storing large amounts of data as documents in JSON format.

Download the Python 3 installer for your platform at python.org. Python is a popular programming language which can be useful for automating processes and handling data.

Install Pymongo Python packages using this command: **“pip install pymongo”.** This is the recommended tool for working with MongoDB using Python.

Install Dash for Python packages using this command: **“pip install dash”.** Dash is used to build the dashboard and is a suitable library for building data apps with Python.

Install other necessary Python data visualization libraries such as Numpy, Pandas, and Matplotlib.

Install Jupyter Notebooks using this command: **“pip install Jupyter”.**

Download the Project\_Two\_Dashboard zip file containing the code for the Dash Web Application, the Grazioso Salvare PNG icon, and the CRUD.py file.

## Usage (CRUD library)

### Code Example

*from pymongo import MongoClient*

*from bson.objectid import ObjectId*

*class AnimalShelter(object):*

*def \_\_init\_\_(self, USER, PASS):*

***# Connection Variables***

**# Update these fields with your personalized information**

***#USER =* ‘*aacuser’***

***#PASS = 'SNHU1234'***

*HOST = '.com'*

*PORT = 00000*

*DB = 'AAC'*

*COL = 'animals'*

***# Initialize Connection***

*self.client = MongoClient('mongodb://%s:%s@%s:%d' % (USER,PASS,HOST,PORT))*

*self.database = self.client['%s' % (DB)]*

*self.collection = self.database['%s' % (COL)]*

***# Create Function***

*def create(self, data):*

***# If data exists, create entry in database and return true***

*if data is not None:*

*self.database.animals.insert\_one(data)*

*return True*

***# Else throw exception and return false***

*else:*

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

*return false*

***# Read Function***

*def read(self, data\_lookup):*

***# If data exists, search database and return list***

*if data\_lookup is not None:*

*data = list(self.database.animals.find(data\_lookup))*

*return data*

***# Else throw exception and return empty list***

*else:*

*raise Exception("Read failed")*

*return []*

***# Update function***

*def update(self, data\_to\_update, update\_to):*

***# If data exists, search database and update***

*if data\_to\_update is not None and update\_to is not None:*

*data = self.database.animals.update\_many(data\_to\_update, update\_to)*

*return data.modified\_count* ***# return num objects modified***

*else:*

*raise Exception("Update Failed")*

***# Delete function***

*def delete(self, data\_delete):*

***# If data exists, search database and delete***

*if data\_delete is not None:*

*data = self.database.animals.delete\_many(data\_delete)*

*return data.deleted\_count* ***# return num objects deleted***

*else:*

*raise Exception("Delete Failed")*

Simply put, the create, read, update, and delete functions check whether the user has input a valid query. If the data does exist then the create function will add the data to the database, the read function will retrieve information from the database, the update function will update entries in the database, and the delete function will delete entries from the database. If the data is non-existent, then exceptions will be thrown. The init function simply connects the user to MongoDB using the provided credentials and information. One challenge I faced when using this script was with the formatting of the credentials. When testing, I used ‘AAC’ rather than ‘aac’, which does not exist as a database. When using this script be careful to use the correct formatting in these fields to ensure a connection.

### Tests (CRUD Library)

**# Import CRUD module**

import CRUD

from CRUD import AnimalShelter

**# Test create function**

data\_create = { "breed" : "Dachshund", "name" : "Samson"}

test\_result = AnimalShelter('aacuser', 'SNHU1234').create(data\_create)

print("Data Created:",test\_result)

**# Test read function**

data\_lookup = { "breed" : "Dachshund", "name" : "Samson"}

test\_result = AnimalShelter('aacuser', 'SNHU1234').read(data\_lookup)

print(test\_result)

**# Test invalid read query**

data\_lookup = { "breed" : "Fake Breed"}

test\_result = AnimalShelter('aacuser', 'SNHU1234').read(data\_lookup)

print(test\_result)

**# Test update function**

data\_to\_update = {"breed" : "Dachshund", "name" : "Samson"}

update\_to = {"$set": {"name" : "Sam"}}

test\_result = AnimalShelter('aacuser', 'SNHU1234').update(data\_to\_update, update\_to)

print("# of objects updated: ", test\_result)

**# Test delete function**

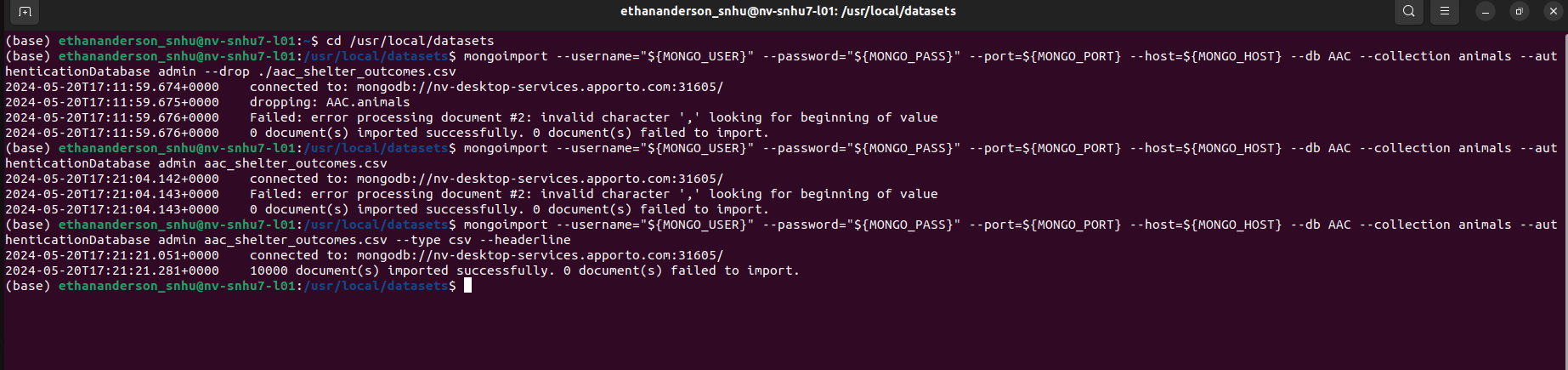
data\_delete = {"breed" : "Dachshund", "name" : "Sam"}

test\_result = AnimalShelter('aacuser', 'SNHU1234').delete(data\_delete)

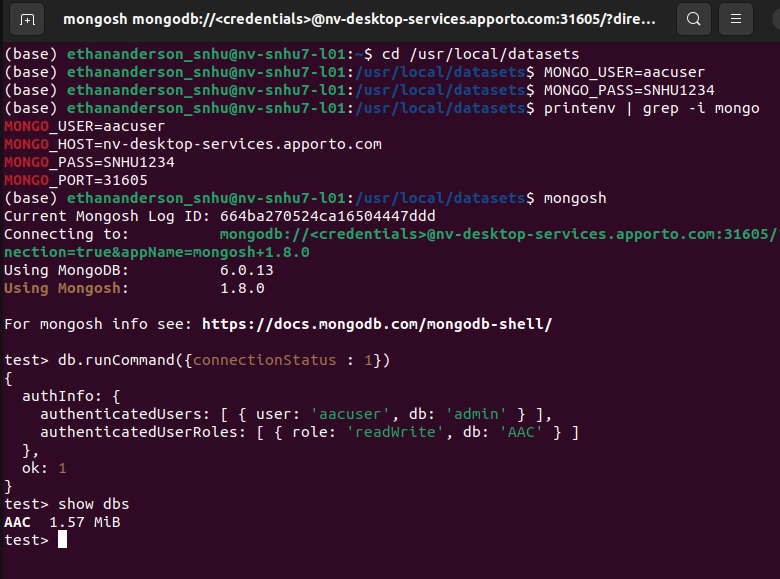
print("# of objects deleted: ", test\_result)

Both the update and delete test functions will return the number of entries which were updated or deleted. The create function will return true if an entry was created, and the read function will return the result of the query in list format.

### Screenshots (CRUD and MongoDB)

****Import .csv to MongoDB**

**Login to MongoDB with Created User Account**

****

## Initialize MongoDB Connection/ Authenticate Username and Password

## 

**Create and Read Functions**

## 

## Update and Delete Functions

## 

## CRUD Module Import and Create/Read Test Functions

## 

## Read, Update, and Delete Test Functions

## 

**Dash Web Application**

The Dash web application includes a data table, pie chart, and geolocation map. Data can also be filtered by different rescue types including Water Rescue, Mountain Rescue, and Disaster Rescue. All this data is filtered using pre-written queries, which are passed into the CRUD Python script.

The dashboard is designed to be user friendly, and includes features such as pagination, scrolling, sorting, selection, and editing. MongoDB is essential for managing the data displayed on this dashboard, as it can store large amounts of data and handle complex queries. There are many different libraries such as Pymongo, Dash, Pandas, etc. for managing and visualizing data from MongoDB to be displayed in a web application.

When running the web application ensure that the CRUD library is imported and that your MongoDB credentials are correct to ensure a connection to the database.

**Usage (Dash Web Application)**

**Read From MongoDB Database**

df = pd.DataFrame.from\_records(db.read({}))

**Create Radio Item**

html.Div([

        html.Label("Filter by..."),

        dcc.RadioItems(id = "filter-type",

                       options = [

                           {"label" : "Water Rescue", "value" : "Water Rescue"},

                           {"label" : "Mountain Rescue", "value" : "Mountain Rescue"},

                           {"label" : "Disaster Rescue", "value" : "Disaster Rescue"},

                           {"label" : "Reset", "value" : "Reset"}],

                       value = "Reset"),

    ]),

**Dashboard Customizations**

**# Enable Single Row Selection**

row\_selectable = 'single',

**# Select First Row by Default**

selected\_rows = [0],

**# Limit Number of Rows Displayed**

page\_size = 10,

**# Enable Pagination**

page\_action = 'native',

**# Enable Sorting (Alphabetical)**

sort\_action = 'native',

**# Scrollbar**

style\_table = {'overflowX' : 'auto'},

**# Editable**

editable = True,

**# Fields to Filter Data**

filter\_action = 'native'

**Example Query**

water\_rescue\_query = {

        "breed" : {"$in" : [

            "Labrador Retriever Mix",

            "Chesapeake Bay Retriever",

            "Newfoundland"]},

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

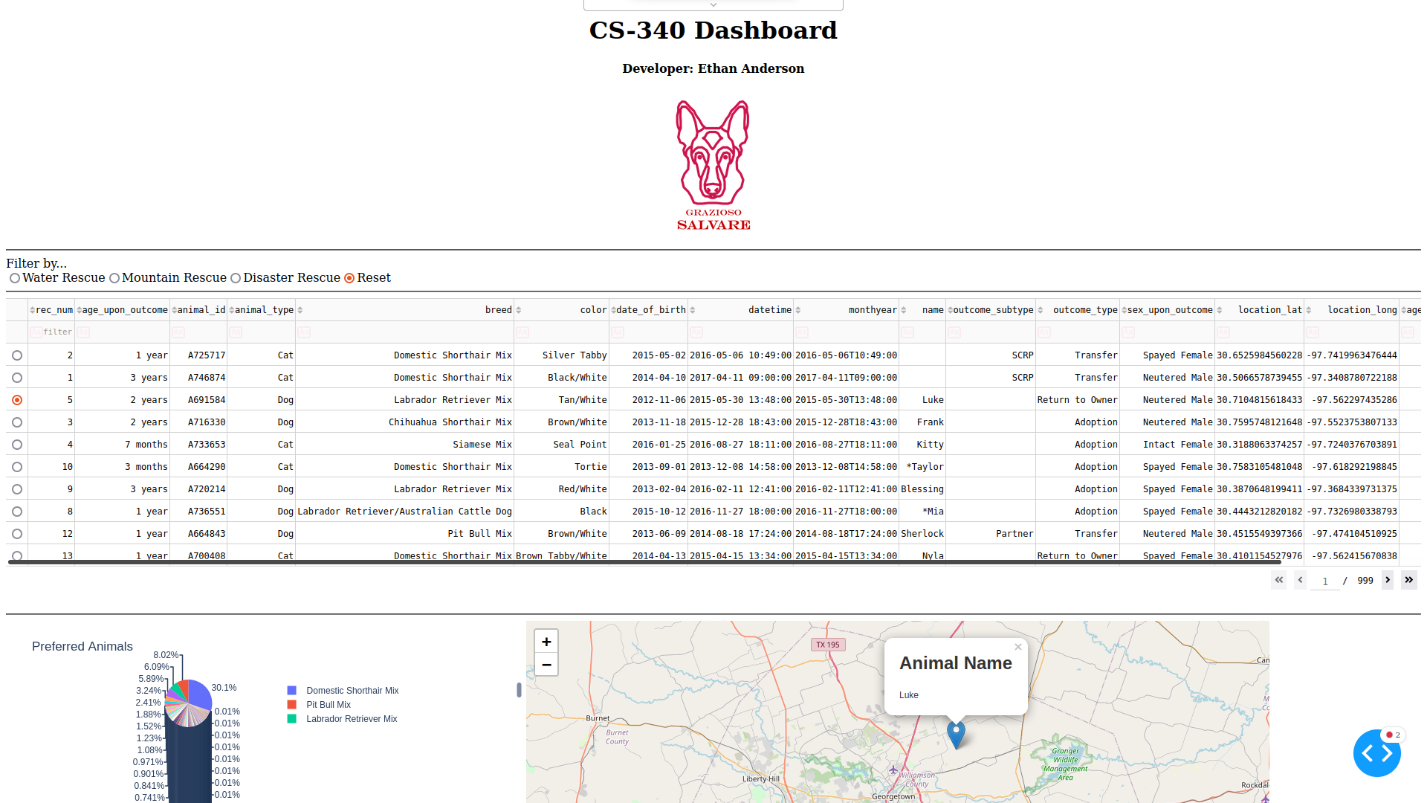
        "age\_upon\_outcome\_in\_weeks" : {"$gte" : 26, "$lte" : 156}

    }

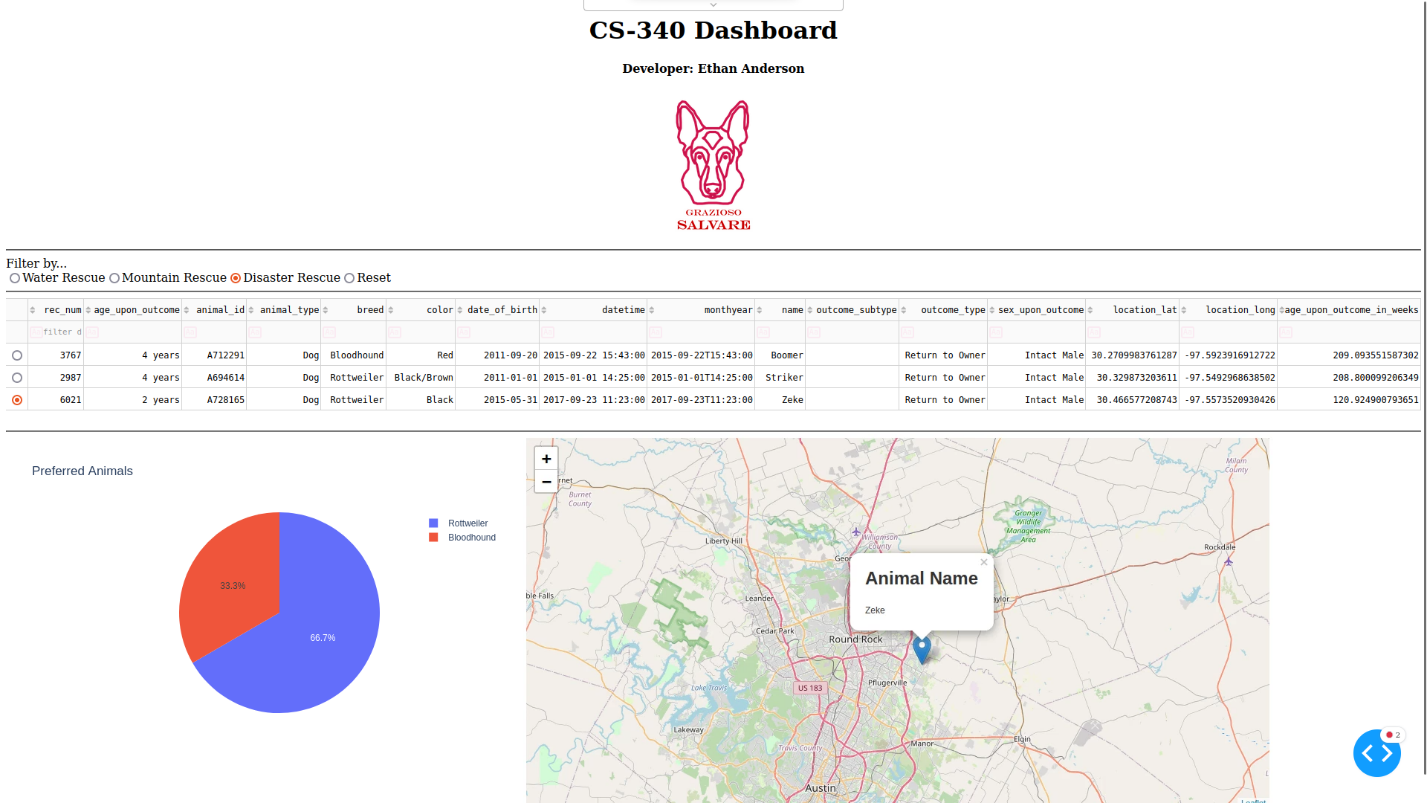
Upon successfully running the Dash application you will see this message: “**Dash app running on** [**http://127.0.0.1:xxxx/**](http://127.0.0.1:xxxx/)**”**

**Screenshots (Dash Web Application)**

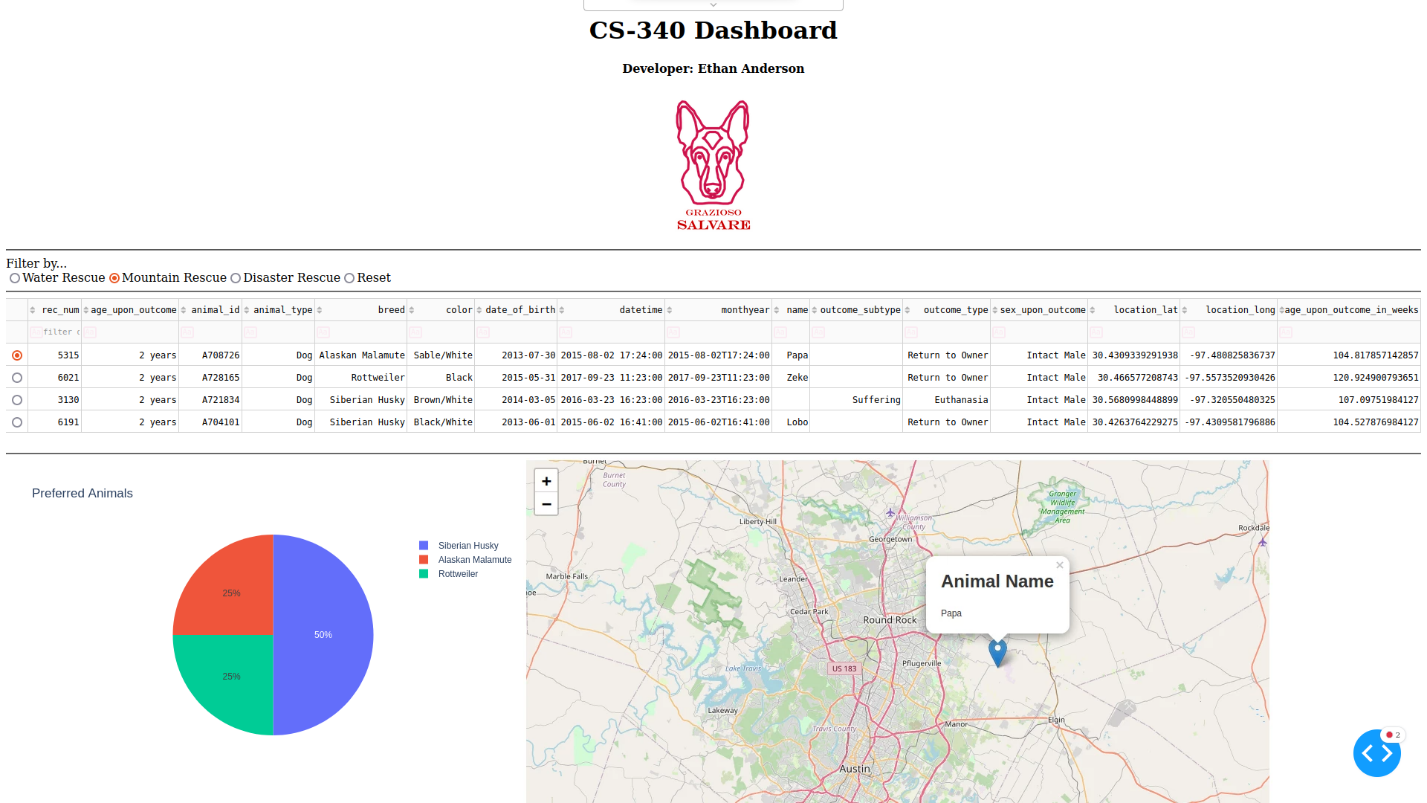
**Dashboard Reset Filter**

****

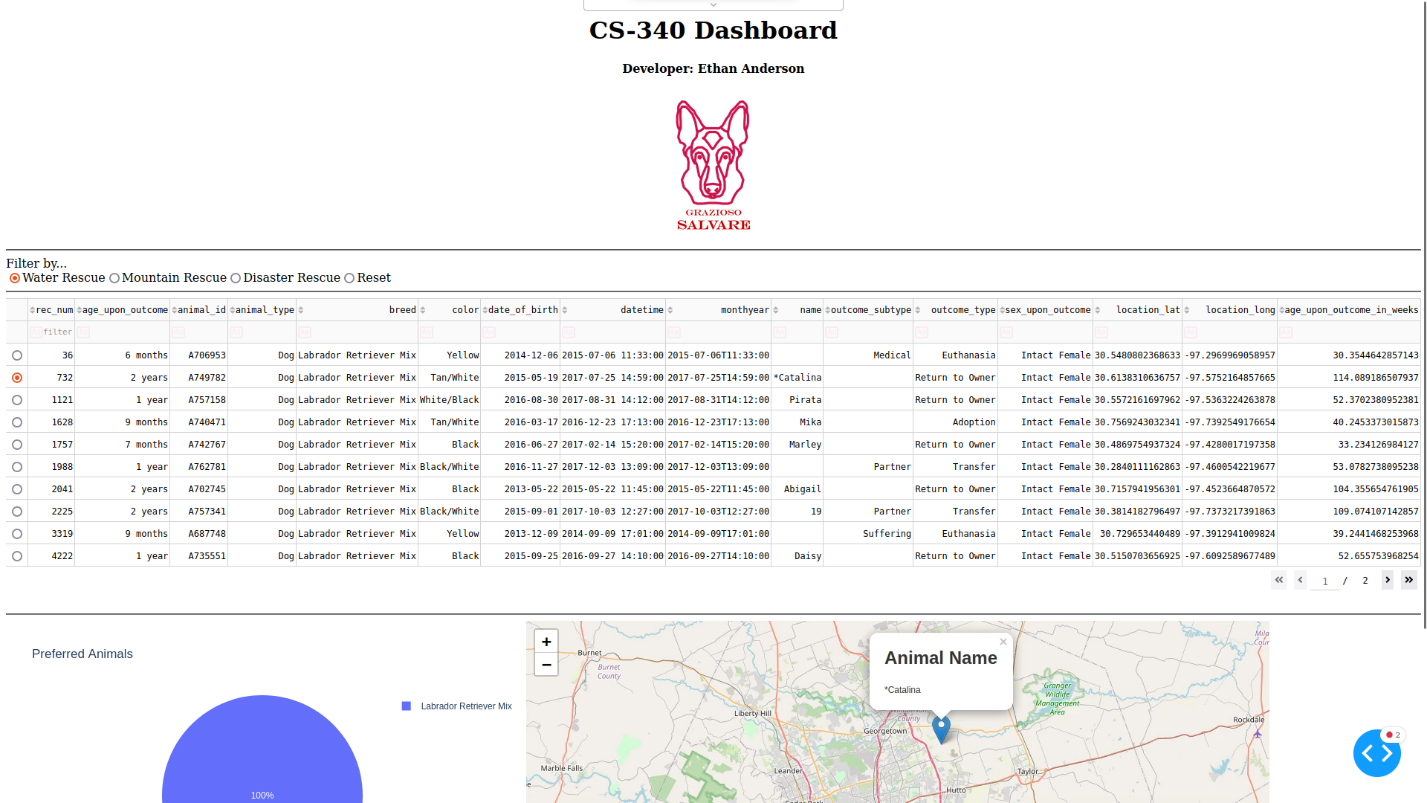
**Dashboard Disaster Rescue Filter**

****

**Dashboard Mountain Rescue Filter**

****

**Dashboard Water Rescue Filter**

****

## Contact

Your name: Ethan Anderson