# CS 340 README CrudDriver and Frontend Dashboard

## About the Project/Project Title

CrudDriver is a Python module that allows you to connect to a MongoDB database to create, read, update, and delete data through member functions. This uses Python dictionaries as the data type for MongoDB documents, which are usually JSON objects. There is also a dashboard frontend that connects to CrudDriver to retrieve documents with certain filters. It uses Dash as an HTML framework to display the data in various ways

## Motivation

Grazioso Salvare animal shelter wanted an easier way to find animals fit to be trained for search-and-rescue. By default, MongoDB must be interacted with through the shell, which would require a lot of typing for this specific use. This project bridges that gap by allowing users to enter the database information once, instance the library, and use simple functions to create, read, update, or delete documents. Furthermore, this project includes a frontend as well. It is a web server in the form of a python terminal file (.ipybn) that runs in Jupyter. This connects directly to the library to retrieve filtered information in the form of a data table, a map, and a pie chart. This was created to allow users with no experience in programming to read the database and easily compile information. It was decided to make the project open source so that anyone can use and modify the code to their specific cases.

## Getting Started

**Refer to Usage section for examples of the code**

**Without frontend script (.ipybn)**

To use this module, you must change the Connection Variables at the beginning of the script to connect to your own database. This supports user authentication, with username and password variables that can be changed. Then, you must import and instance the module into either a Python script, or a Python terminal. From there you can now access the member functions, which take dictionaries as input.

**With frontend script**

Change the Connection Variables at the beginning of the frontend script to connect to your own database. Run the frontend script in Jupyter Notebooks, click the link it gives as output, and read the database however you like.

**Information and Behavior**

* The library's member functions, excluding create(), will read, update, or delete all objects that match the first argument. It is necessary to be aware of this, and recommended to use unique identifiers as the first argument when updating or deleting one document.
* The delete function is the only function that takes two dictionaries as input, with the first being the search criteria, and second being the fields to be updated.
* The member functions use pyMongo’s find, insert\_one, update, and delete functions. These support indexing, which can be set up in the Mongo shell.
* The member functions will return different values. create() returns a boolean, read() returns a list, and update() and delete() return the number of documents that were updated or deleted.
* The dictionaries are Python’s equivalent of the JSON objects MongoDB uses.
* The frontend dashboard displays all documents, or those within a certain criteria
* The database must be in the form of the one this was made for to work without modification
* The dashboard allows users to search any field of the documents
* There are also radio buttons which each correspond to a specific set of traits in dog breeds
* The frontend will also display a pie chart based on breed frequency, and a map to show the location of the animal selected in the data table above
* The dashboard uses dash as an HTML framework for python. This allows the webpage to be programmed entirely in python using app.layout = *‘ dash html element’*. The functionality can be in the same document using @app.callback(Output(*‘element-id’: ‘field’*), [Input(*‘element-id’: ‘field’*),...]), which watches for updates to the element and field in the Input field, and outputs to the element in the Output field.
* Plotly is used to display the pie chart, and compile the data from the list that’s on the screen
* As a reminder, there are Connection variables at the beginning of each script. These are used to connect to the server and database. The script will not work without changing them for your specific database. Change them in the backend script to use the library with your own cases. Change them in the frontend script if using the frontend.

**Challenges**

* pyMongo’s find function returns a cursor object, and not the document itself. Therefore it is necessary to retrieve the object the cursor has selected, which is done simplest by Python’s list function. A loop could also be used.
* The choice of using pyMongo’s multi-query functions (find(), update\_many(), delete\_many()) rather than single query functions (find\_one(), update\_one(), delete\_one()) was decided by the fact that users may want to act on multiple documents, even at the possibility of updating the wrong documents. This is why it is important to use unique identifiers when only updating or deleting one document.
* The given code was returning the data and columns to the DataTable when a radio button was clicked. Updating the columns broke the code and prevented the documents in the query from displaying in the table. Returning only the data fixed the problem.

## Installation

This requires Python, a MongoDB server, and multiple Python packages. Refer below for the packages. These can be downloaded from python.org, and mongodb.com, and in the terminal, respectively. This project also uses the bson.objectid module, because MongoDB uses these for unique identifiers, the base64 module for rendering images, and the os module. You will also notice that notebook needs to be installed as a python package. This is also a terminal application that will run the frontend script (.ipybn).

**Required Python packages:**

* notebook
* jupyter\_dash
* dash\_leaflet
* dash
* plotly
* numpy
* pandas
* matplotlib

**Reason for usage**

MongoDB was chosen as the database because it supports many useful features, such as indexing, subdocuments, and Python compatibility. Dash is also compatible with Python, and provides a simple way to program a frontend, so that means the entire project could be in one language to reduce the footprint and cost. Plotly, which is used for graphing, is compatible with Dash, and, as seen by this project, doesn’t require much code to integrate. Jupyter was chosen to run the server on since it is a lightweight and well featured terminal for Python.

## Usage

### Code Example of interacting with CrudDriver

*from cruddriver import CrudDriver*

*crud\_driver = CrudDriver()*

*crud\_driver.create( {“name” : “Cameron”, “Age” : 22} )*

*crud\_driver.create( {“name” : “Cameron”, “Age” : 12} )*

*print(crud\_driver.read( {“name” : “Cameron”} ))*

*print(crud\_driver.update( {“name” : “Cameron”}, {“name” : “Cam”} )*

*print(crud\_driver.read( {“name” : “Cam”} ))*

**Output:**

[{‘\_id’ : ObjectId(‘…’), ‘name’ : ’Cameron’, ‘Age’ : 22}, {‘\_id’ :ObjectId(‘...’), ‘name’ : ‘Cameron’, ...}]

2

[{‘\_id’ : ObjectId(‘…’), ‘name’ : ’Cam’, ‘Age’ : 22}, {‘\_id’ :ObjectId(‘...’), ‘name’ : ‘Cam’, ...}]

### Tests

*from cruddriver import CrudDriver*

*crud\_driver = CrudDriver()*

***Create()***

*print(crud\_driver.create(None))*

**Output:** False

Does not create document when there is no data

*print(crud\_driver.create({}))*

**Output:** False

Does not create document when there is no data

*print(crud\_driver.create( {“name” : “Cameron”, “Age” : 22} ))*

**Output:** True

Successfully creates document

***Read()***

*print(crud\_driver.read(None))*

**Output:** []

Returns empty list and does not search with no criteria

*print(crud\_driver.read({}))*

**Output:** []

Returns all documents in database

*print(crud\_driver.read( {“name” : “Michael”} ))*

**Output:** []

Returns empty list when no matches are found

*print(crud\_driver.read( {“name” : “Cameron”} ))*

**Output:** [{‘\_id’ : ObjectId(‘…’), ‘name’ : ’Cameron’, ‘Age’ : 22}]

Returns list of matching documents

***Update()***

*print(crud\_driver.update(None))*

**Output:** 0

Returns 0 and does not search with no criteria

*print(crud\_driver.update({}))*

**Output:** 0

Returns 0 and does not search with no criteria

*print(crud\_driver.update( {“name” : “Michael”} ))*

**Output:** 0

Returns 0 when no matches are found

*print(crud\_driver.update( {“name” : “Cameron”} ))*

**Output:** 1

Returns number of modified documents

***Delete()***

*print(crud\_driver.delete(None))*

**Output:** 0

Returns 0 and does not search with no criteria

*print(crud\_driver.delete( {“name” : “Michael”} ))*

**Output:** 0

Returns 0 when no matches are found

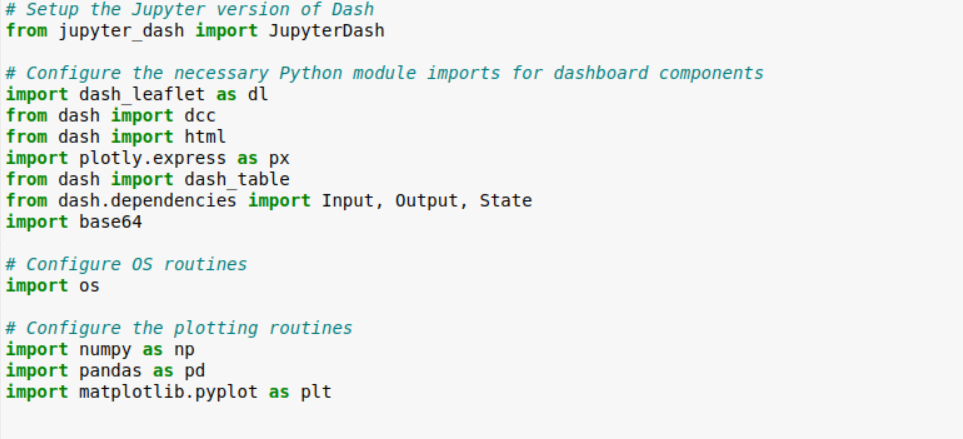
*print(crud\_driver.delete( {“name” : “Cameron”} ))*

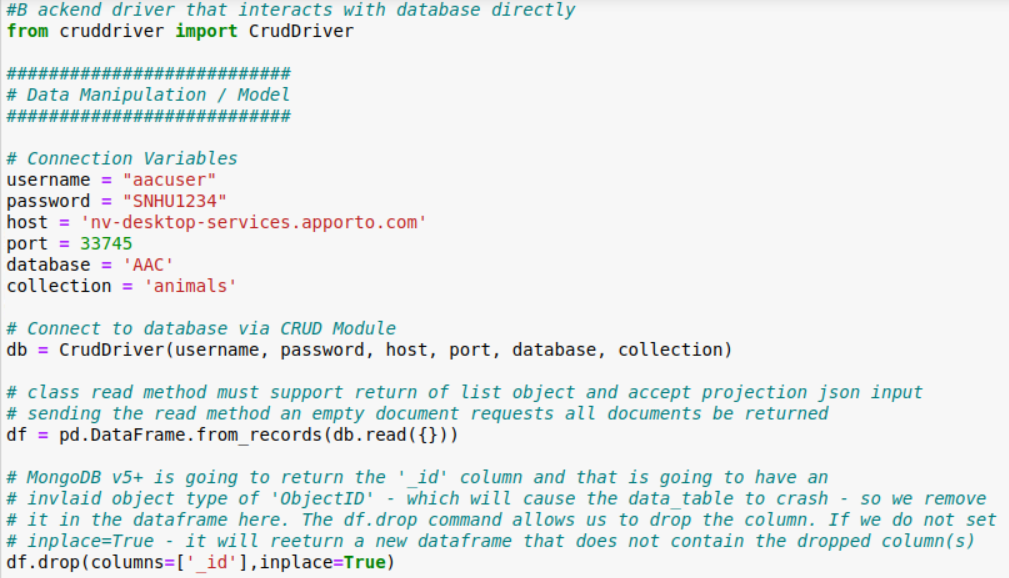
**Output:** 1

Returns number of document deleted

### Code Example Screenshots of frontend script

Example of connecting to library and connecting the library to a database





Displaying database as a table with dash

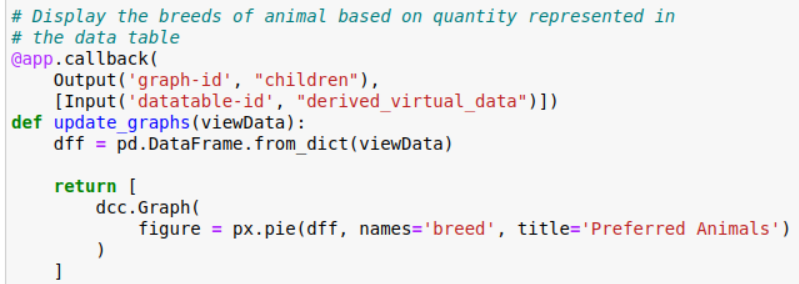




…



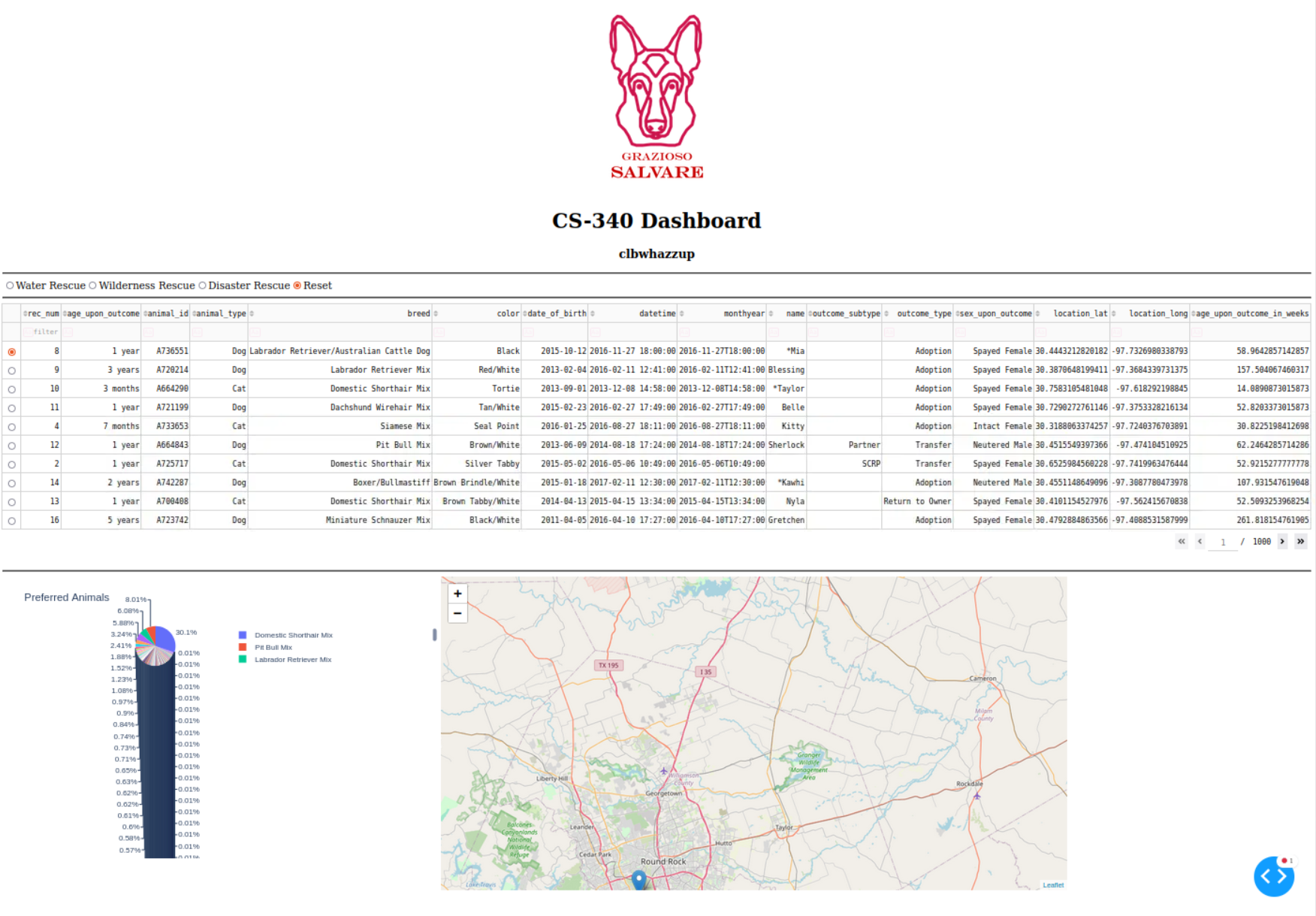
Displaying the database as a pie chart of the frequency of breeds

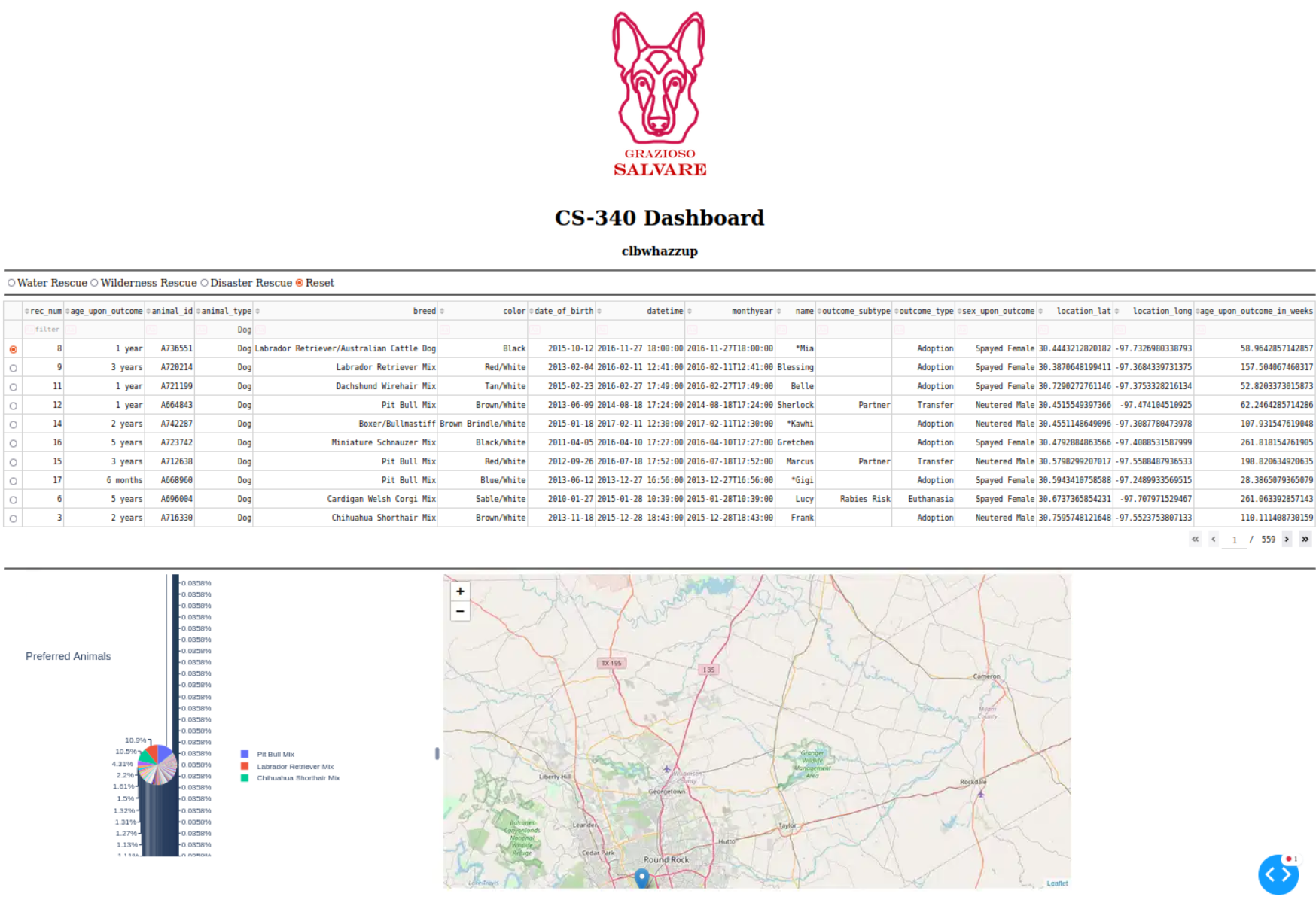


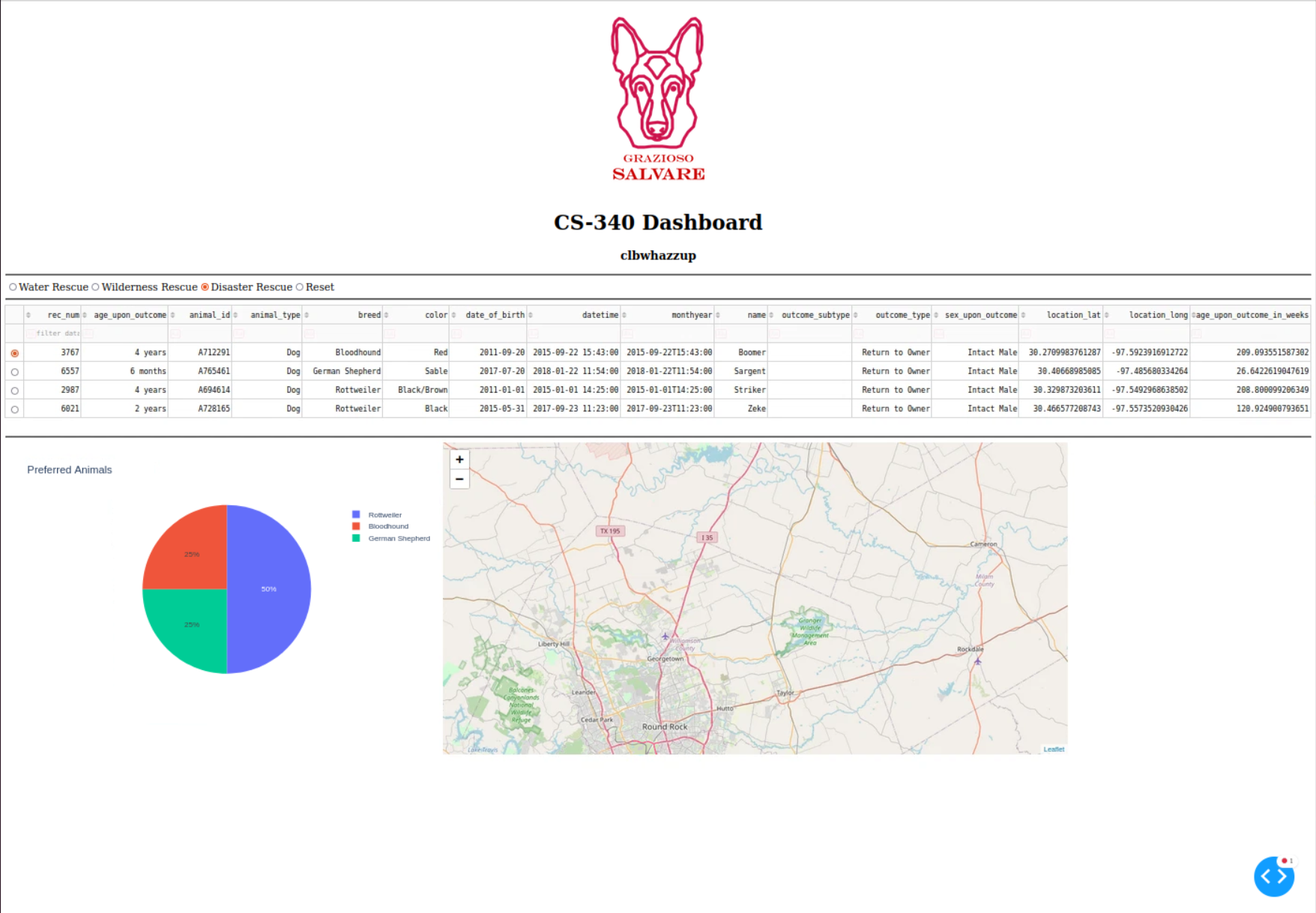
Showing the backend library used in the frontend

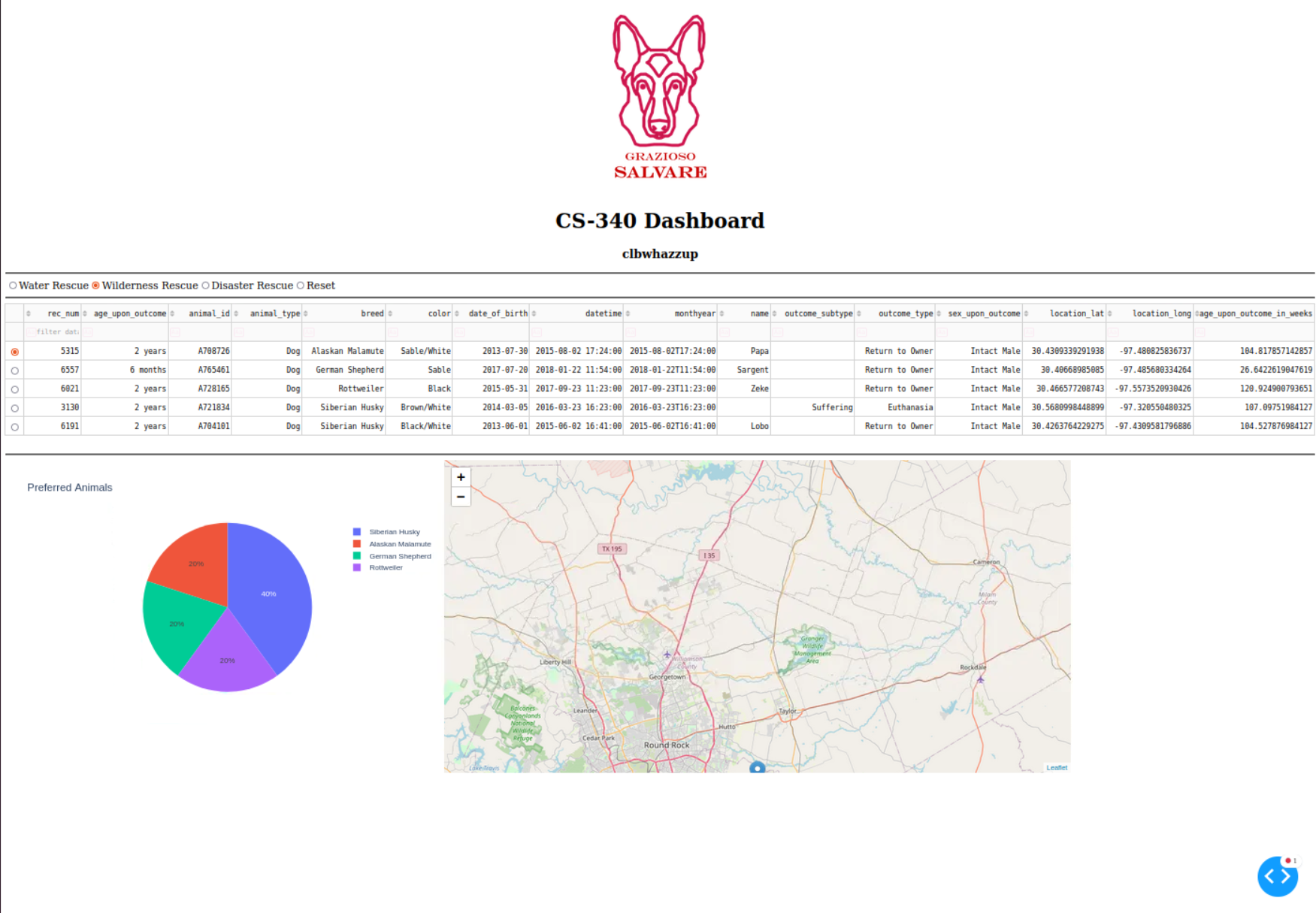


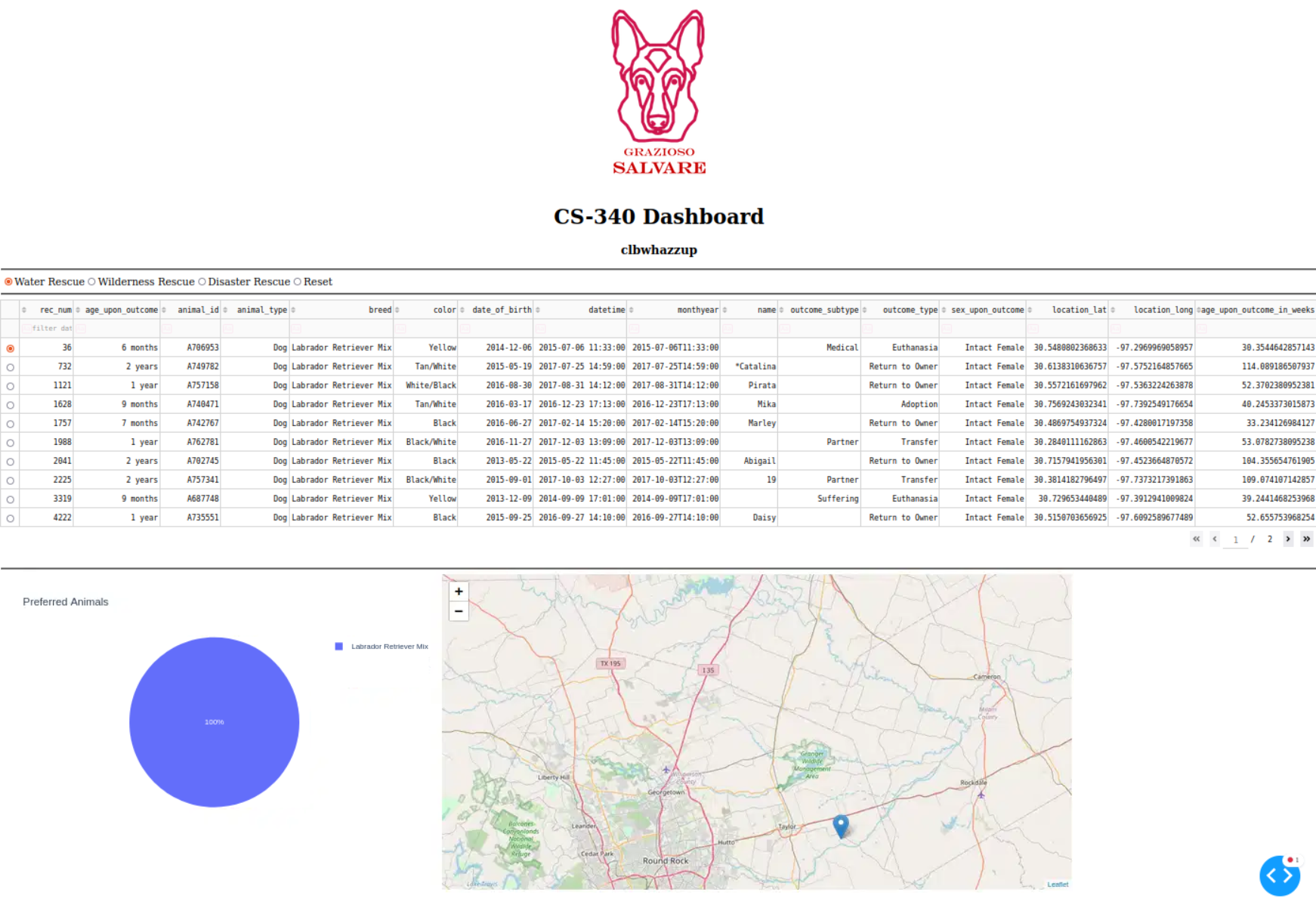
**Screenshots of frontend dashboard with filtering options**











## Contact

Your name:

Cameron Binda