**CS 340 Module Seven - Project Two README**

Kemal Cankurt

SNHU

​​Course Number:​ CS 340

David Ostrowski

# About the Project/Project Title

This time, we took things to the next level! After building the Python CRUD module in Project One, we’ve now developed a full interactive dashboard for Grazioso Salvare. This dashboard lets users easily explore and filter animal outcome data, especially focusing on dogs that are ideal for different rescue missions like Water Rescue, Mountain or Wilderness Rescue, or Disaster Tracking. Users may also find additional filtering for precise results such as animal type, breed, outcome type (adoption or missing etc.), or animal’s name.

# Motivation

The goal was to help Grazioso Salvare staff quickly spot the right animals for training by using a slick UI. Users can filter by breed, outcome, name, and type. Bonus? There are radio filters for specific rescue categories! The backend is powered by MongoDB and our trusty animal\_shelter.py module from Project One.

# Tools and Technologies Used

To build the dashboard, I used a mix of Python tools and web technologies that made everything run smoothly together:

* Dash by Plotly: Dash is a powerful Python framework for building interactive web apps. It combines Flask on the backend with React.js on the frontend, which makes it super convenient for Python developers who don’t want to dive deep into HTML/JS. Dash handled the whole layout and logic of the dashboard, from dropdowns to charts and maps. I liked how easy it was to plug in components and connect callbacks for live updates.
  + More about Dash: <https://dash.plotly.com>
* MongoDB: This is the database I used to store and retrieve animal outcome data. I chose MongoDB because it’s a flexible NoSQL database that works well with JSON-style documents, which fits perfectly with Python's dictionaries. It was also easy to connect to using pymongo, and the document structure made it simple to query and filter based on user selections.
  + MongoDB info: <https://www.mongodb.com>
* Pandas: I used this to transform and filter the data I pulled from MongoDB. It's a go-to library for handling tabular data and worked well to shape the data for the table and charts.
* Plotly Express: This was used to build the pie chart showing the distribution of preferred animal breeds. It’s part of the Plotly ecosystem and works naturally with Dash, which helped a lot.
* Dash Leaflet: For the interactive geolocation map, I used dash\_leaflet, which let me drop pins on a live map based on latitude and longitude from the dataset. This really added to the visual side of things.
  + Dash Leaflet: <https://www.dash-leaflet.com>
* Bootstrap 5 (via CDN): To make the dashboard look cleaner and more organized, I included Bootstrap 5’s stylesheet. It helped me create responsive layouts, cards, and form controls with minimal CSS work.
* JupyterDash: Since the project runs in a Jupyter Notebook, jupyter\_dash was the tool that let me launch and test the web app inside the notebook environment.
* Other resources used:
  + Dash documentation: <https://dash.plotly.com/dash-core-components>
  + MongoDB with Python (pymongo): <https://pymongo.readthedocs.io>
  + Bootstrap: <https://getbootstrap.com>

These tools worked well together to give me a smooth development experience and made it possible to create a fully functional, dynamic dashboard all in Python.

# Getting Started

To get a local copy up and running, follow these steps:

1. Ensure MongoDB is installed and running, or access the provided MongoDB instance hosted in Apporto.
2. Import the dataset using the following command:

| mongoimport --host nv-desktop-services.apporto.com --port 30122 -u aacuser -p SNHU --authenticationDatabase admin --db AAC --collection animals --type csv --file /usr/local/datasets/aac\_shelter\_  outcomes.csv --headerline |
| --- |

1. Set up user authentication (aacuser with readWrite access to AAC database).
2. Use the animal\_shelter.py module to connect and interact with the database.



# Installation

The following tools and libraries are used in this project:

* Python – Main programming language
* pymongo – Python driver for MongoDB
* mongosh – MongoDB shell to test connections and data
* Jupyter Notebook – Used to test and demonstrate the module

Install dependencies using pip:

| pip install pymongo |
| --- |

# Usage

Import and use the module in a Jupyter Notebook as follows:

import importlib

import animal\_shelter

importlib.reload(animal\_shelter)

from animal\_shelter import AnimalShelter

# Instantiate the AnimalShelter class

shelter = AnimalShelter()

# Create operation: Insert a sample document

sample\_animal = {

"name": "Fluff",

"animal\_type": "Dog",

"breed": "Belgian Shepherd",

"age\_upon\_outcome": "2 years",

"outcome\_type": "Adoption"

}

insert\_result = shelter.create(sample\_animal)

print(f"Insert Successful? {insert\_result}")

# Read operation: Query based on animal name

query\_result = shelter.read({"name": "Fluff"})

print(f"Query Result: {query\_result}")

## How does this module work?

This module uses the "pymongo" library, the official Python driver for MongoDB. It supports MongoDB's query language and integrates well with Python's object-oriented features, allowing easy interaction with collections through simple syntax and structures like dictionaries and lists.

The AnimalShelter class connects to the MongoDB database and provides the following methods:

* create(data): Inserts a new document into the animals collection and returns "True" if successful.
* read(query): Returns a list of documents matching the provided query using "find()."
* update(query, new\_values): This function updates documents matching the query and returns the modified document count.
* delete(query): This function deletes documents that match the query and returns the number of deleted documents.

Each method has basic error handling, with "try/except" blocks that log messages when operations fail.

## Code Example

def create(self, data):

"""

Inserts a document into the MongoDB collection.

:param data: dict - key-value pairs representing the document

:return: True if insert is successful, else False

"""

if data:

try:

result = self.collection.insert\_one(data)

return True if result.inserted\_id else False

except PyMongoError as e:

print(f"Insertion failed: {e}")

return False

else:

raise ValueError("Cannot insert empty data")

def read(self, query):

"""

Queries documents matching the key/value pair in MongoDB.

:param query: dict - key-value pair to search

:return: list of matching documents, or empty list

"""

try:

result\_cursor = self.collection.find(query)

results = list(result\_cursor)

return results

except PyMongoError as e:

print(f"Read failed: {e}")

return []

## How to use Dashboard

Using the dashboard is simple and intuitive. At the top, you'll find the Rescue Type section with four radio button options: Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking, and Reset. Selecting any of these will filter the data automatically based on matching dog breeds and animal type, according to the client’s rescue use cases.

Below that, you have more detailed filtering options: dropdowns for Animal Type, Breed, and Outcome Type, plus a text input to search by name. All these filters work together using AND logic, so you can mix and match to narrow things down exactly how you want. The name search field uses debounce behavior, meaning it updates when you press Enter or click outside the field, not on every keystroke to avoid unnecessary refreshes.

Once you apply any filters, the interactive data table updates in real-time. You can scroll, sort, and click rows to select an animal. This selection dynamically updates the two charts below: a pie chart showing breed distribution, and a map showing the selected animal’s geolocation.

The dashboard is fully connected and everything reacts to your input without needing to reload. Just pick your filters, explore the results, and the visuals will keep up with you.

## Challenges and Fixes

While building the dashboard, I ran into a few bumps that needed some extra attention. One of the first issues was how filters were behaving. At first, they were acting like OR conditions instead of AND, so selecting multiple filters was still returning unrelated results. To fix this, I rewrote the filter logic so all selected options combine into a single MongoDB query, ensuring results matched every selected filter.

Another challenge came from callback errors, especially when no data matched a filter. The pie chart and map would crash when trying to render empty or missing values. I added checks to handle empty DataFrames and fallback text when there's nothing to display, which kept everything from breaking.

Styling was also a bit tricky. I wanted the dashboard to be clean and responsive without using custom CSS. I solved this by importing Bootstrap from CDN and using grid classes to properly size dropdowns and layout components.

Lastly, the map wasn't centering correctly on selected animals, but adjusting the center coordinates inside the callback fixed it up. With those issues resolved, the dashboard worked smoothly across all filters and views.

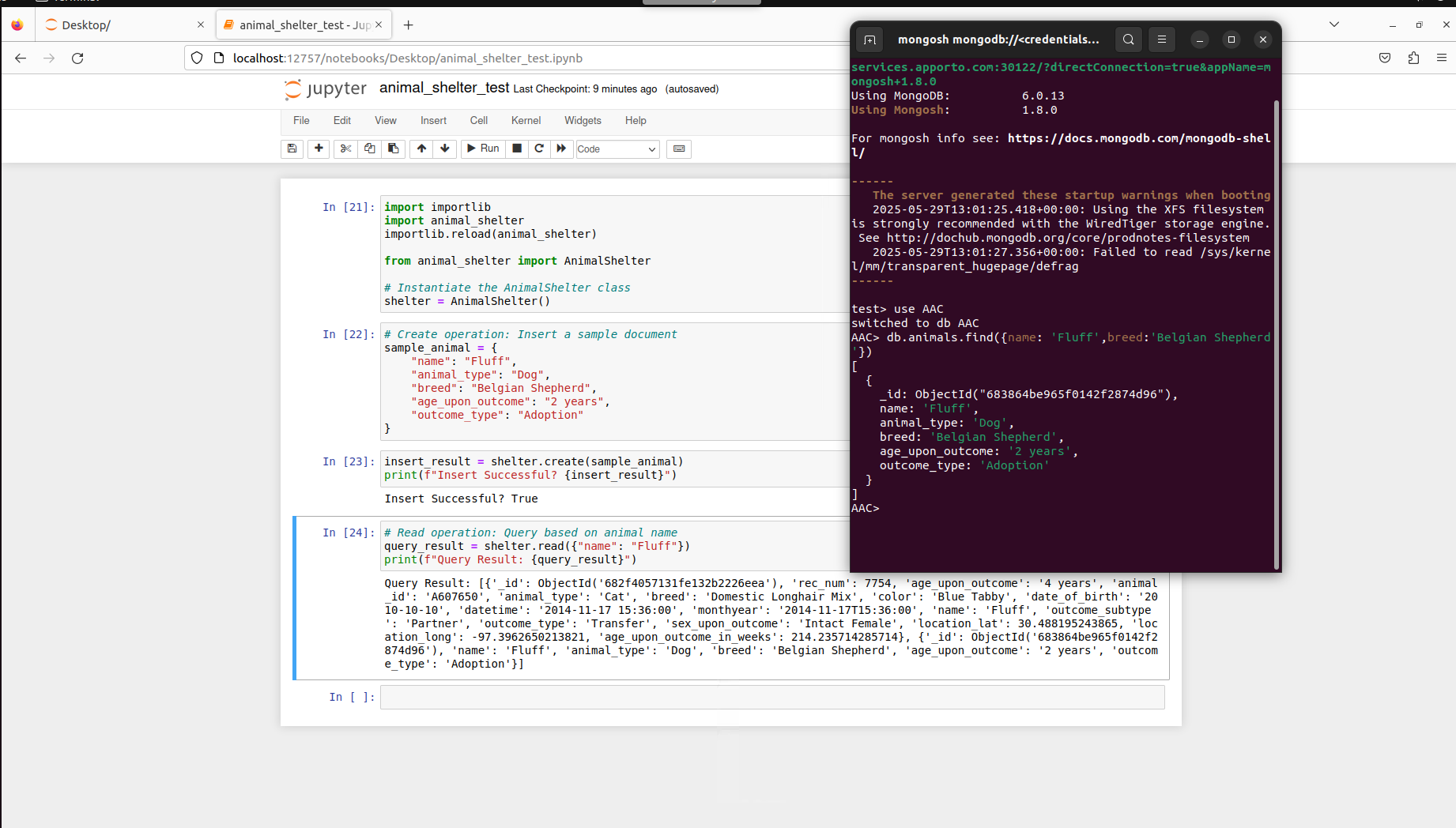
## Tests

I carried out the testing in a Jupyter Notebook, using some sample data to call the create, read, update, and delete methods. It was great to validate the results by checking the printed output and running queries in MongoDB!

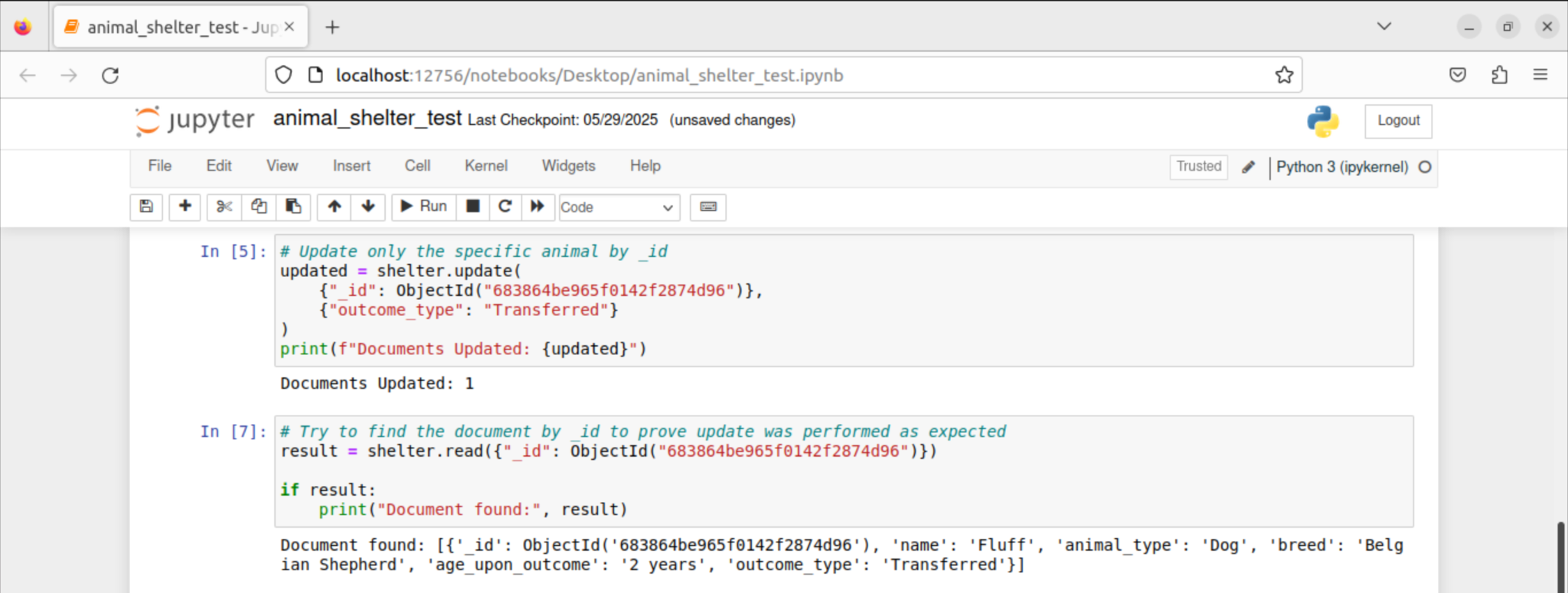
For the dashboard's interface, I tested each of the UI components by running the app and using the filtering options like Rescue Type, Animal Type, Breed, Outcome Type, and Name search. I made sure the filters all work together properly and the data table, map, and chart update in sync. I also confirmed that the reset option brings everything back to the default state. The screenshots included in this project show the filtering steps and help prove the dashboard works as expected visually too.

## Screenshots

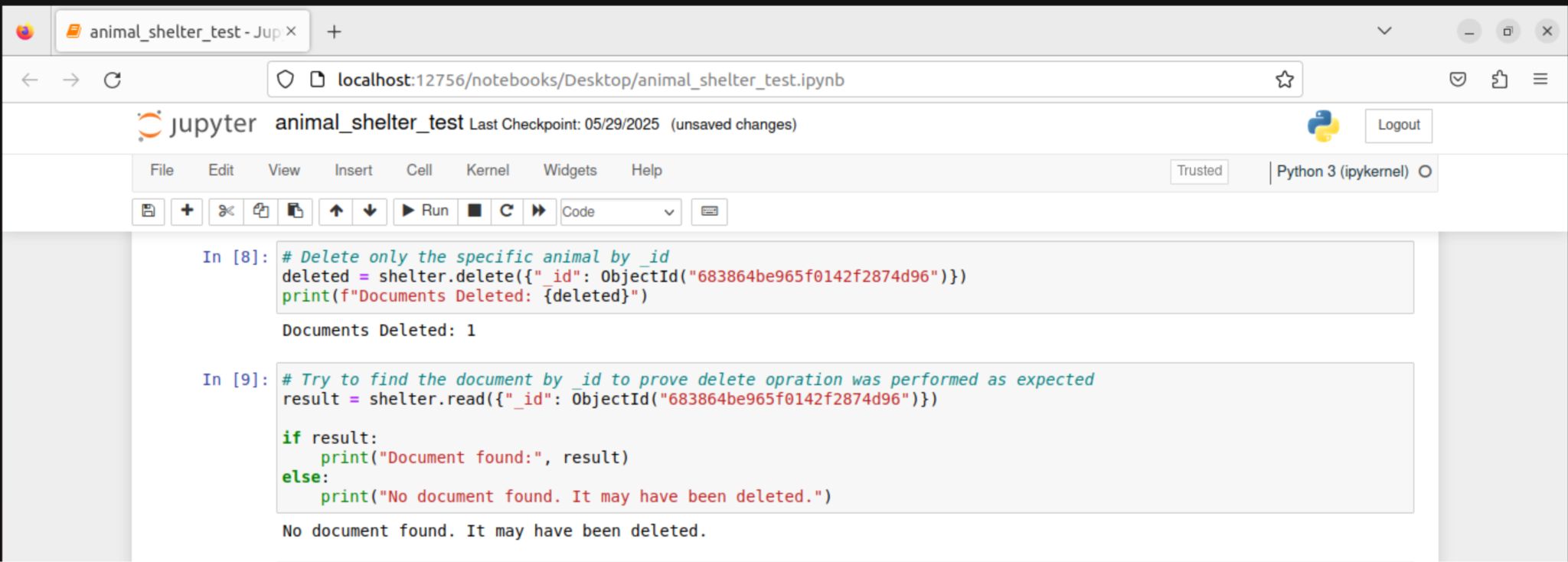
### Create and Read

****

### Update

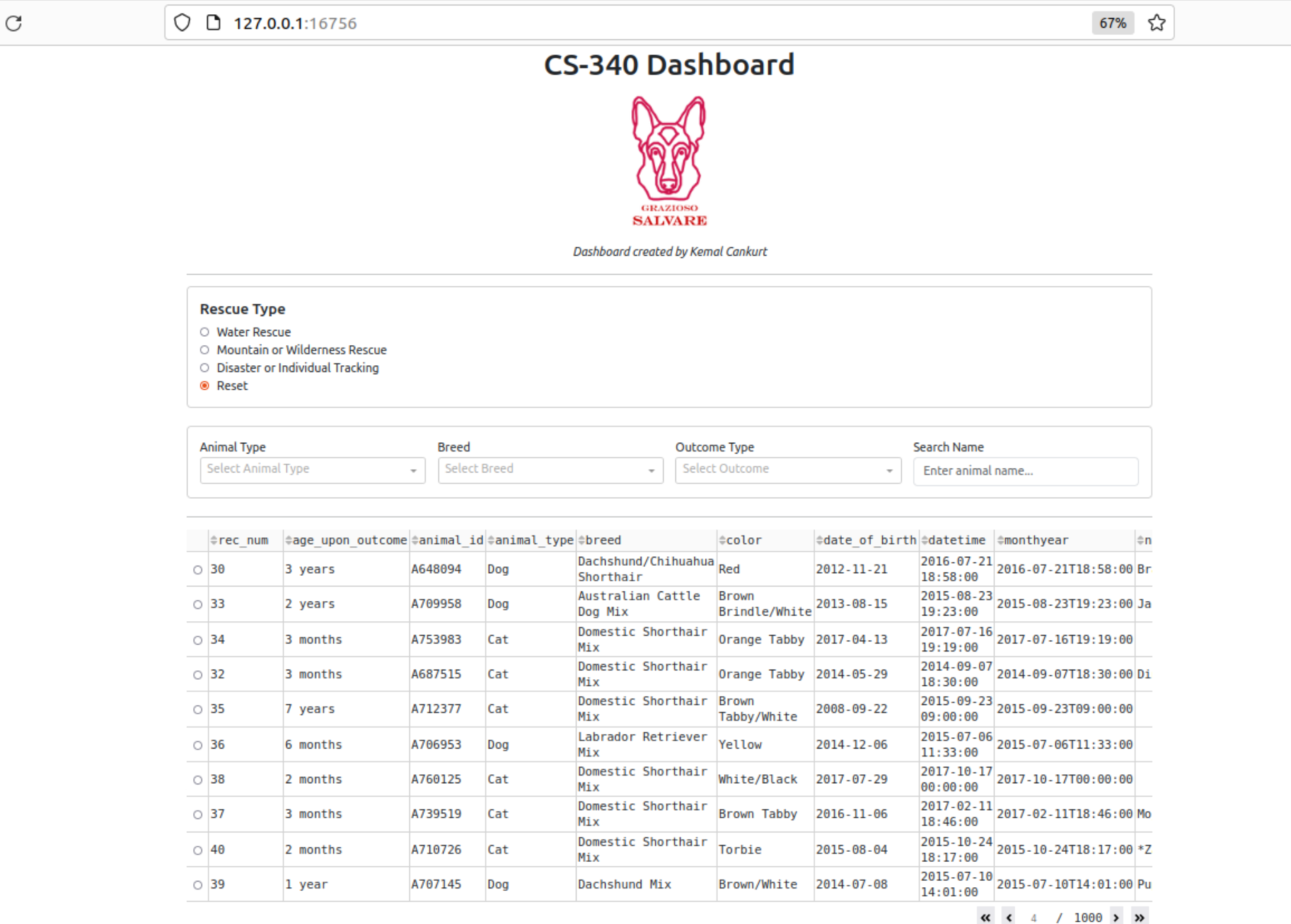


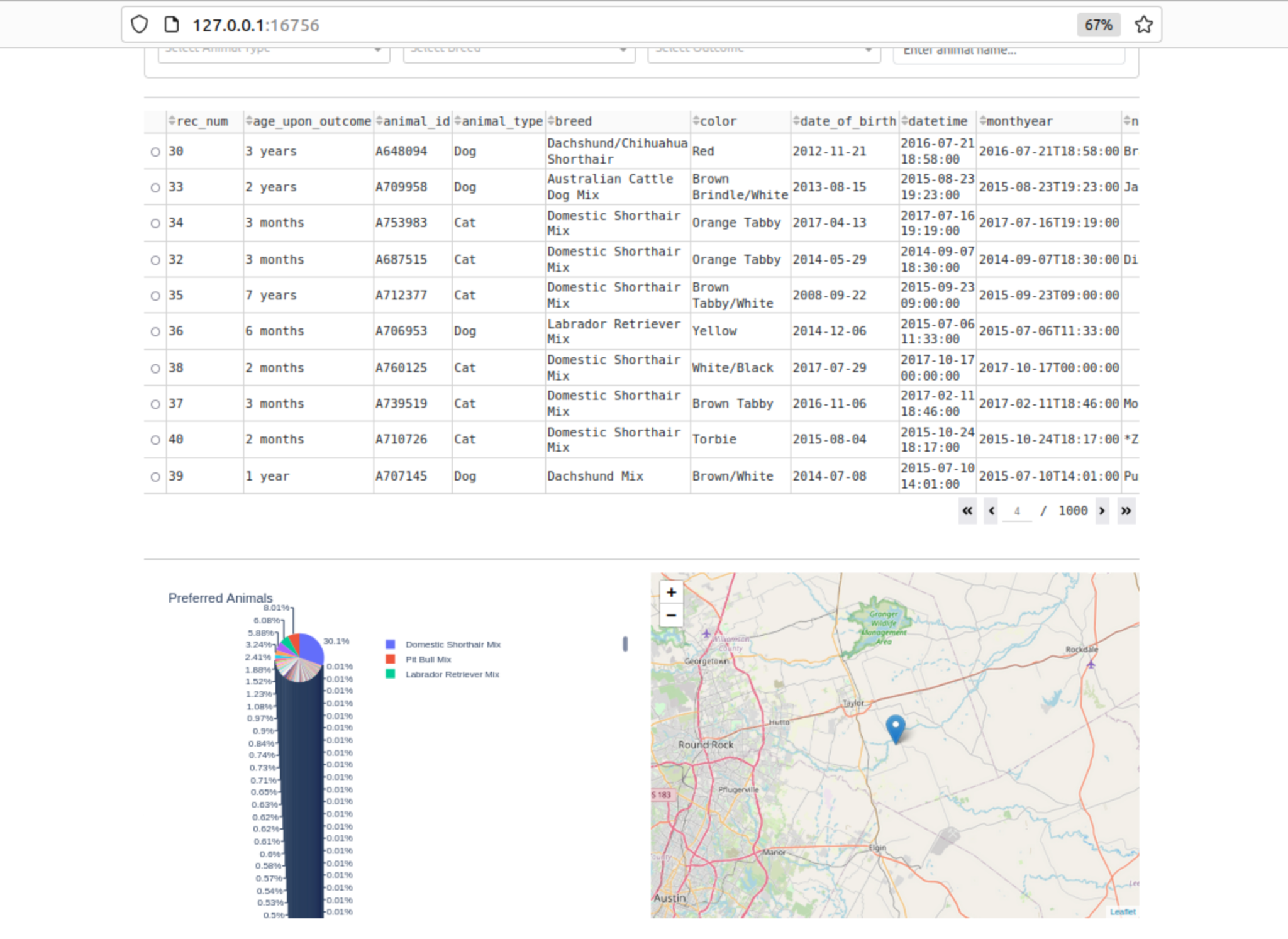
### Delete



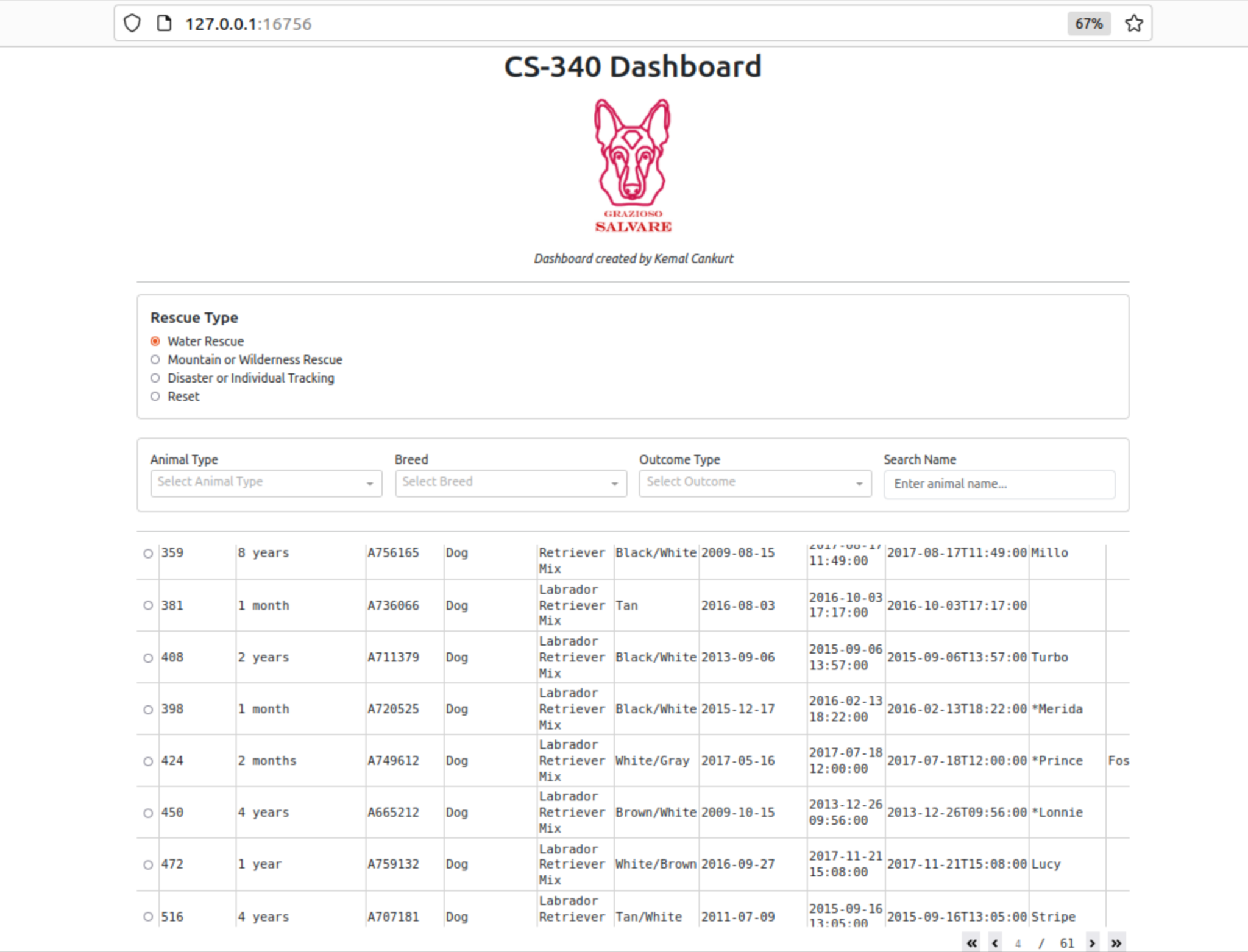
### Dashboard

#### Default View

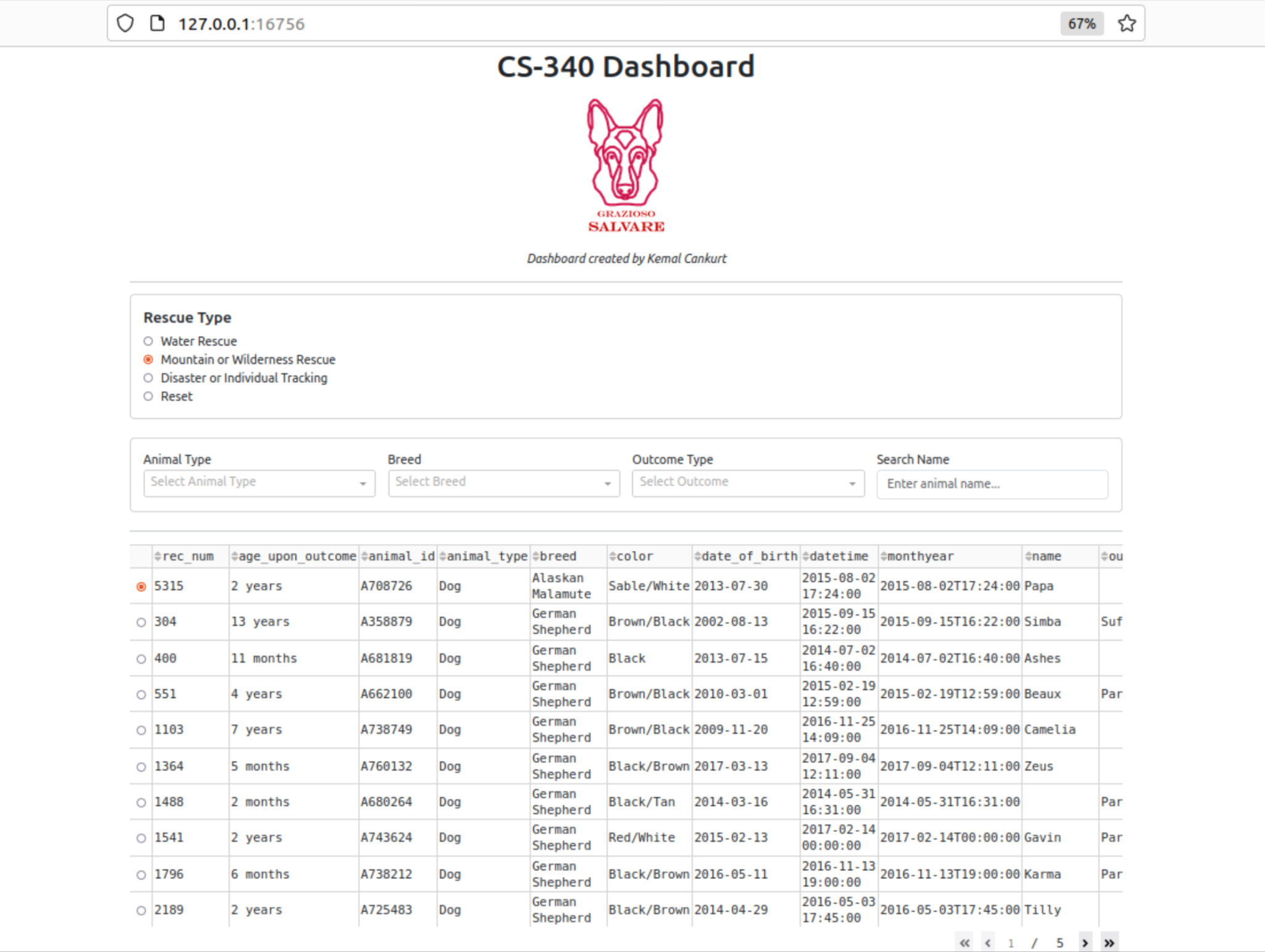




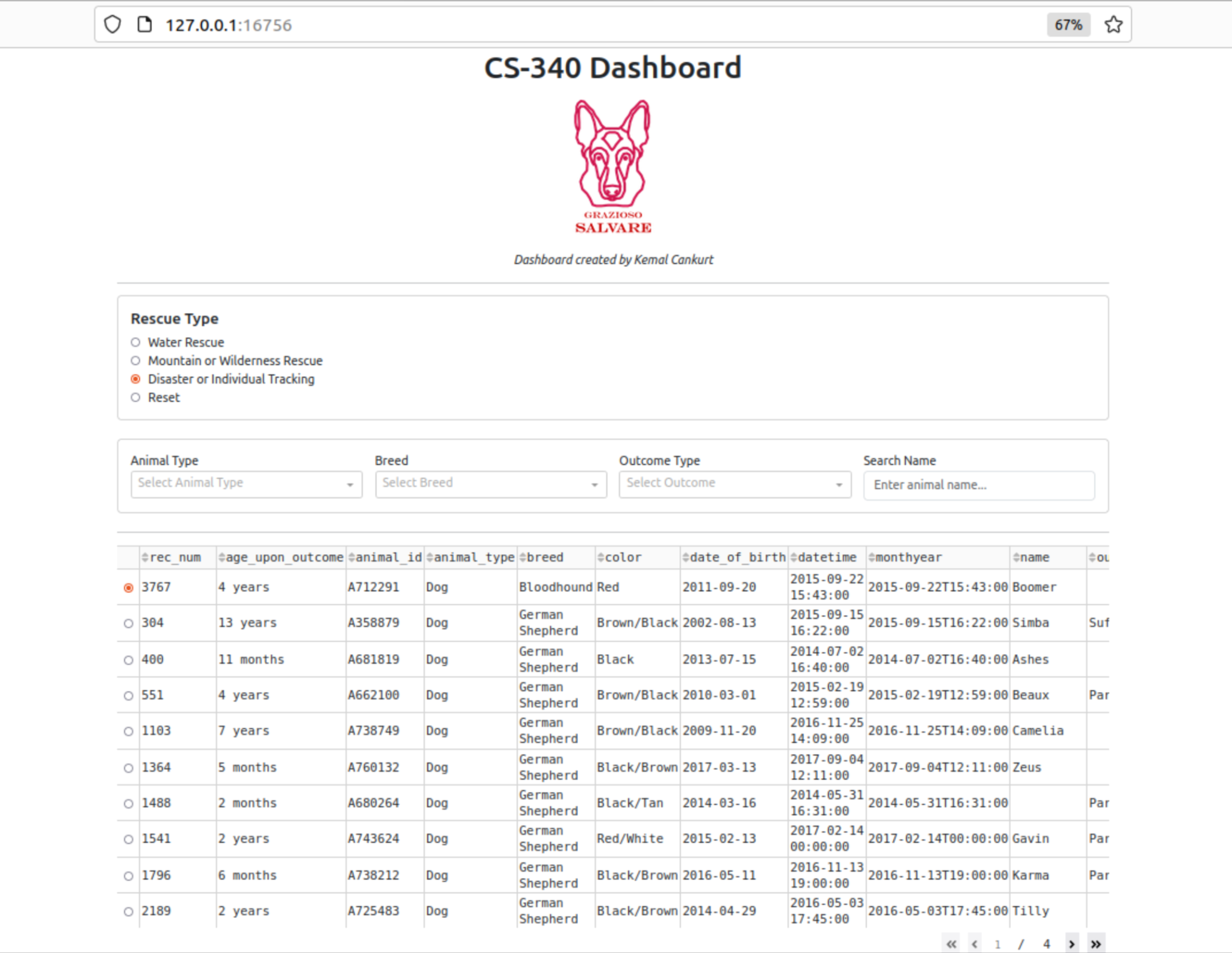
#### Water Rescues



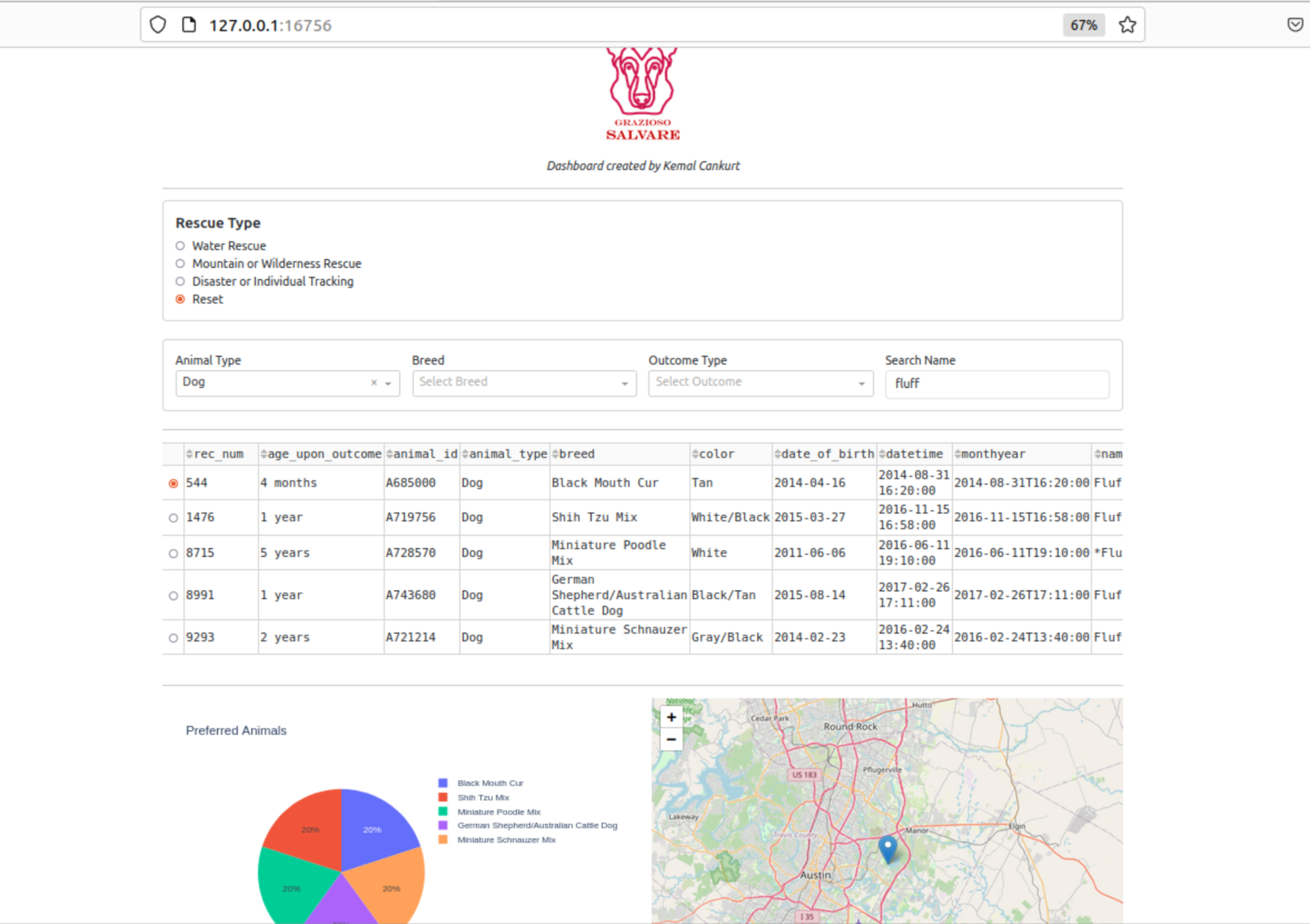
#### Mountain or Wilderness Rescue



#### Disaster or Individual Tracking



#### Search by Name & Animal Type



# 

# Contact

Kemal Cankurt

[kemal.cankurt@snhu.edu](mailto:kemal.cankurt@snhu.edu)