**CS-340 Project 2 README**

**-About the Project:**

This project is being developed for an organization called Grazioso Salvare, which is an international rescue animal training company. Grazioso Salvare finds dogs around the world that are good candidates for search/rescue training and trains them to help humans and other animals. Some dogs are better when it comes to training potential, which is why Grazioso Salvare came to us for help with filtering and finding such dogs in a database.

This project is now fully complete, minus some minor details and refinements. The application connects a database to a UI, through a Python middle layer to allow for data to be found easily in the database. The database used in this model is a MongoDB database that stores info about dogs and cats in animal shelters. However, other types of databases, such as ones with only dogs can be used as long as they have the correct fields for queries. The UI for the application is deployed through an IPYNB file that utilizes the Dash libraries, which are very useful for UI development in Python. The last part, or the middle layer, is the .Py file which allows for operations to be performed on the database thanks to the PyMongo official library. The .Py file/Python Module is used to read info from the database, which is then used by the IPYNB file to display the data in a chart. The UI allows for the data being read to be filtered based on the type of rescue animal, which will force the UI to update the table. The UI also displays a pie chart and a geolocation map to give users a better idea of where the dogs are and extra info about them.

**-Motivation**

This project, for the most part, was developed in order to meet the needs of the client Grazioso Salvare. They wanted our company to develop an application that made it easier for their users to find potential rescue dogs from the databases that Grazioso Salvare uses. Not only did they want us to develop the back-end code, but they also wanted us to develop some of the front-end code such as the UI. We developed the base functionality of the database including the CRUD operations. We also developed the connection between the database, UI, and CRUD module in order for it all to work and produce the wanted output. Overall, I feel like this project provided me with an opportunity to gain experience working with Python libraries like Dash and Database Systems like MongoDB. I was also motivated to make a web application like this that felt like a very complete project.

**-Getting Started**

* The first step in using this application in a local setup is to install MongoDB and Jupyter Notebooks (or some other type of Python IDE)
* The next step is to enter a command prompt, and start your MongoDB instance. Do not use the start-noauth parameter to start the MongoDB instance, just use mongod\_ctl start.
* Once your MongoDB instance is started, you will need to either create a new dataset or import a dataset by using the appropriate mongo import command.
* After getting your dataset imported, you need to create a few roles for the database such as an admin role and a database user role. You can create these kinds of roles in the database by first accessing the mongo shell by typing mongo in the cmd. Then, you can use the command db.createUser() to create a user type with a set name, password, and role permissions.
* Lastly, restart your MongoDB instance and your new user types should be all good to go. You can check to see if your user types work by logging in to them by starting up the mongo shell with the username. You will then be asked to enter the password you set earlier to complete the login.
* With user roles and the database setup, the next thing you need to do is access a Python IDE such as Jupyter Notebook.
* When you access Jupyter Notebook, you must create a new ipynb file in which you will write code for the UI, and a .Py file that will hold the functions for interacting with the database.
* Once you have your code files made, you can then copy the code from the repository into your files. You may need to change the port number and database name in your \_\_init\_\_ function before the code works correctly. (This is in the Python module/.Py file)
* As for your IPYNB file, you will have to copy the code over from the repository, as it is the final piece in the puzzle for the application to work. You will most likely have to change the name of the imported Python module and possibly the name of the read function being called multiple times in the IPYNB file. You will also have to edit certain text things such as the types of filters that exist. Setting up the code within the IPYNB file will allow for the UI to be displayed whenever the database is online and the IPYNB file is run.
* After this, you should be able to mess around and work with the core of the application. Your database will be connected to the Python Module, which will also be connected to the IPYNB UI file. Thus, you will be able to perform operations on the database from the IPYNB file using a neat interface!

## **-Installation**

The tools you will need to operate the software are:

**MongoDB** – This acts as the main database manager where our databases are stored. It is very useful for creating and managing NoSQL databases. I feel it is important to use it because Python actively supports the use of MongoDB through the PyMongo library and the application needs a way to securely build a database. The installation for MongoDB can be found here: [Try MongoDB Community Edition | MongoDB](https://www.mongodb.com/try/download/community)

**Python IDE** – This will act as the main way of writing, compiling, testing, and running the Python files for the project. There needs to be an easy way to write the Python file while also checking to make sure the code is error free, which makes having a Python IDE necessary. All in all, there are many different Python IDEs out there, so it is really up to personal preference for which one is used. You can install them by visiting the appropriate IDEs website. I used Jupyter Lab for my work, which made it very easy to write code for the UI. I would recommend using Jupyter Lab or Notebook to write your .py and .ipynb files since it is very easy to test and see the results of said tests in Jupyter Lab/Notebook. Just remember that if you use Jupyter Notebook, you can load the running results of code in another tab or window. It helps in case you need to take screenshots.

**Dash Framework –** The Dash Framework is a set of libraries that can be imported to Python projects that work very well when it comes to setting up web UIs. The Dash libraries make it easy to set up apps that work with data by supplying a way for vastly customizable UIs. The UIs developed through Dash are rendered in a web browser which makes them easily accessible and very user friendly. Dash utilizes a fair bit of HTML, so it is important to understand some of the basics behind the HTML language. In the project, Dash is used to build the web UI that interacts with the Python module, which in turn, interacts with the MongoDB database. Personally, I really liked using Dash because it made it very easy to develop the UI for the application. I am semi-proficient with HTML, so some of the Dash statements were very familiar to me. Here is a guide for installation: [Part 1. Installation | Dash for Python Documentation | Plotly](https://dash.plotly.com/installation)

**PyMongo** - This is the officially supported driver for Python by MongoDB. It allows MongoDB to take commands from a Python script/module. It is used in this project as a way for the Python script/module to execute CRUD operations on the MongoDB database. I felt it was best to use this driver because it is the officially supported driver.

**Mongo shell** - The mongo shell is used to access databases and create certain functionalities within them such as indexes or user roles. Mongo shell comes with MongoDB when the MongoDB server installation is installed. For this project, the mongo shell is needed to set up the roles for authentication and authorization.

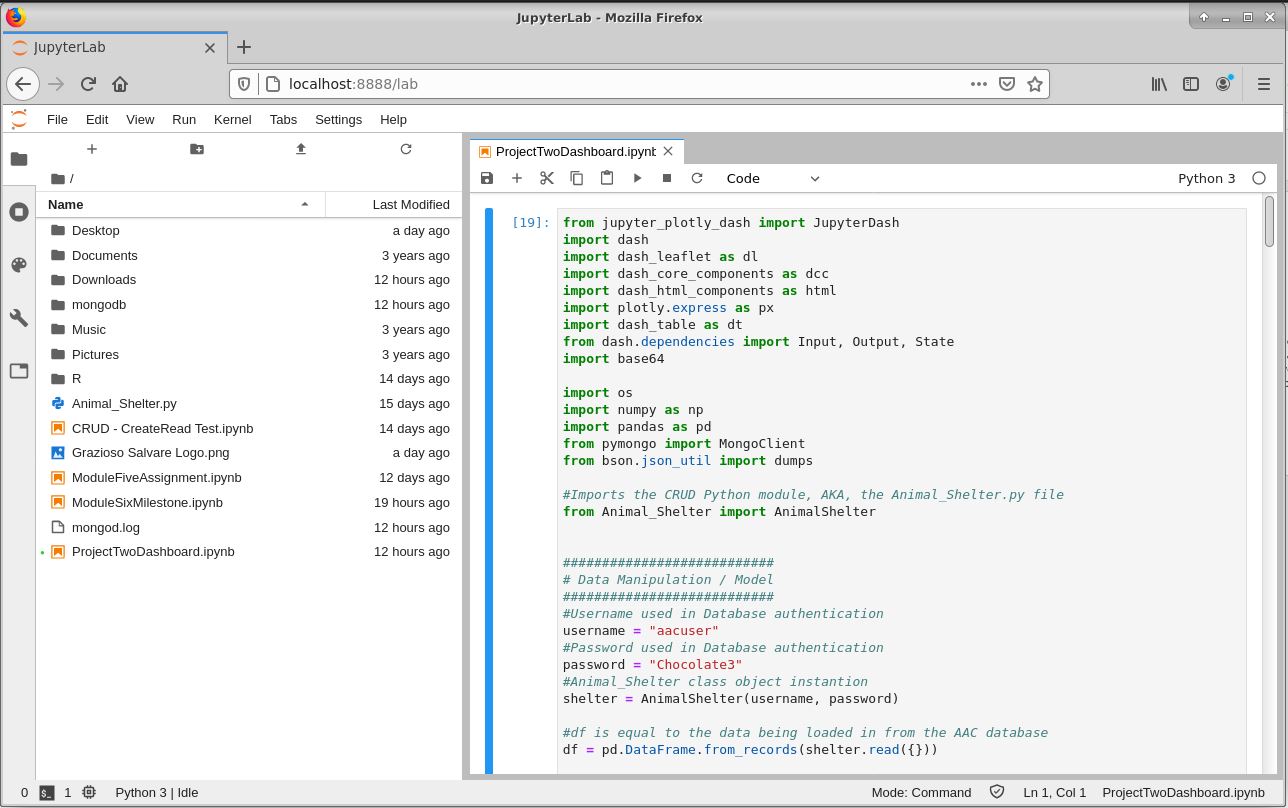
**-Code example**

As was mentioned before, the code for this project connects the MongoDB database to the User Interface stored in the IPYNB file, via a connection from the Python CRUD module. The IPYNB file that acts as the UI allows for the database to be displayed in the UI in the form of a table, while also showing info from the database in the form of a Geolocation map and a Pie chart. The only function from the Python CRUD module that is utilized in the IPYNB UI file is the read function. The read function from the Python CRUD module connects the MongoDB database to the IPYNB UI, allowing for the IPYNB UI to have interactive filtering via specialized read command calls.

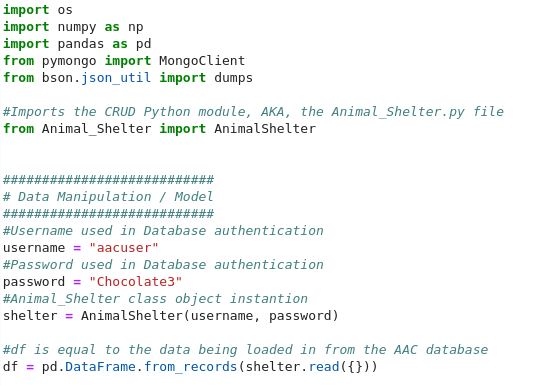
In terms of this project, the code can be used to display an interactive web UI that works with a database to allow users to find rescue dogs for different scenarios. Users would also be able to pinpoint a specified dog’s location via the geolocation map provided by the UI. The functionality of this project can be altered to make it work for other databases, but it will take reconfiguration work in both the IPYNB and .Py files.

Here is an example of the Project working all together to display the interactive UI:

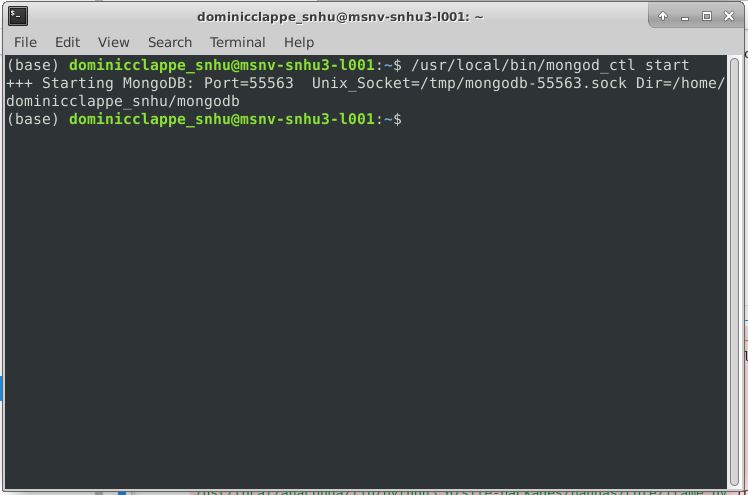
* Once you have the CRUD Python module, the MongoDB database, and the code for the IPYNB file all set up, the next step is to actually run the code to see if the connections worked!
* First, you want to open your Python IDE. In the case of our example, we will be using Jupyter Lab/Notebook.

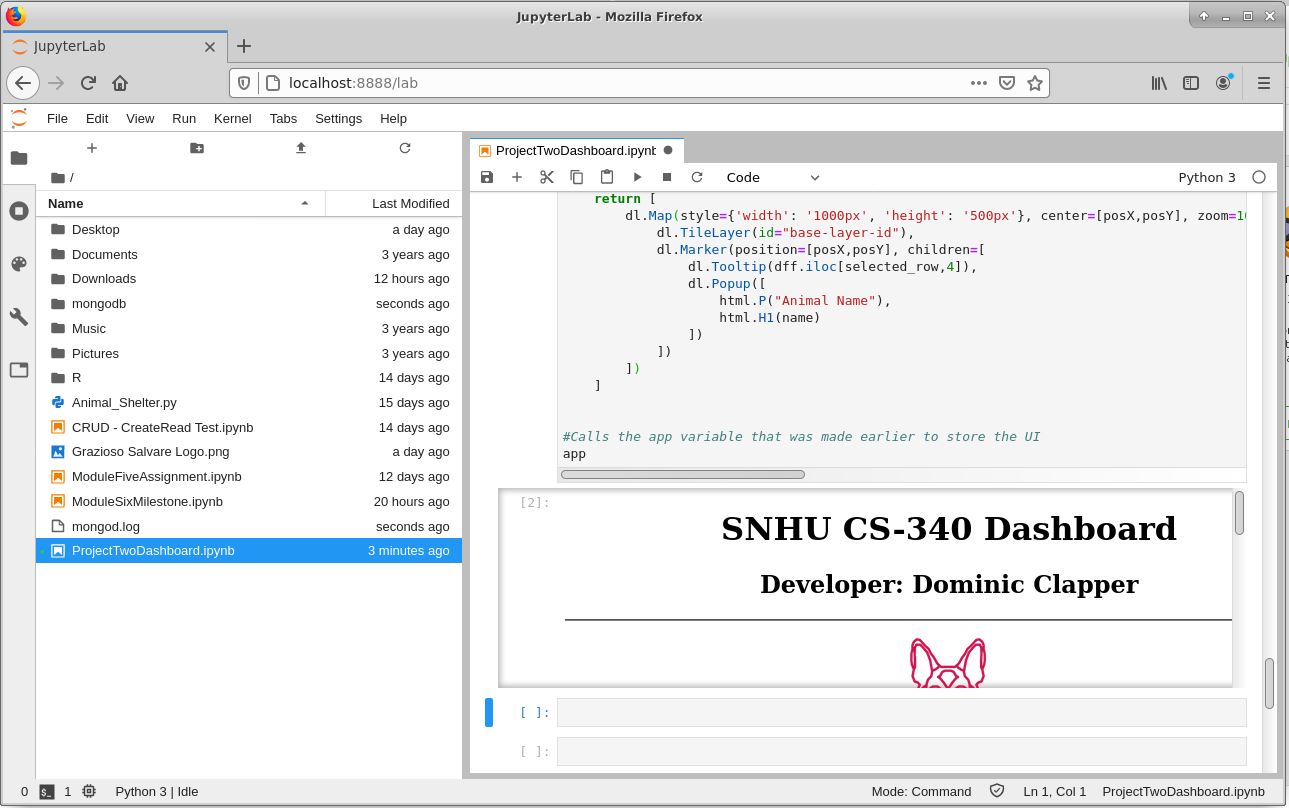


* Once you have your Python IDE open, the next step is to make sure all of your code is present. If you have copied the code provided in the repository, then it should all be there. You may need to change the name of your import of the CRUD Python module and the username and password for the database authentication process.



* With all the code loaded, the next step is to make sure the MongoDB database is open and running. Then, you can click on the code module in the IPYNB file and run the code to see if the UI is displayed!

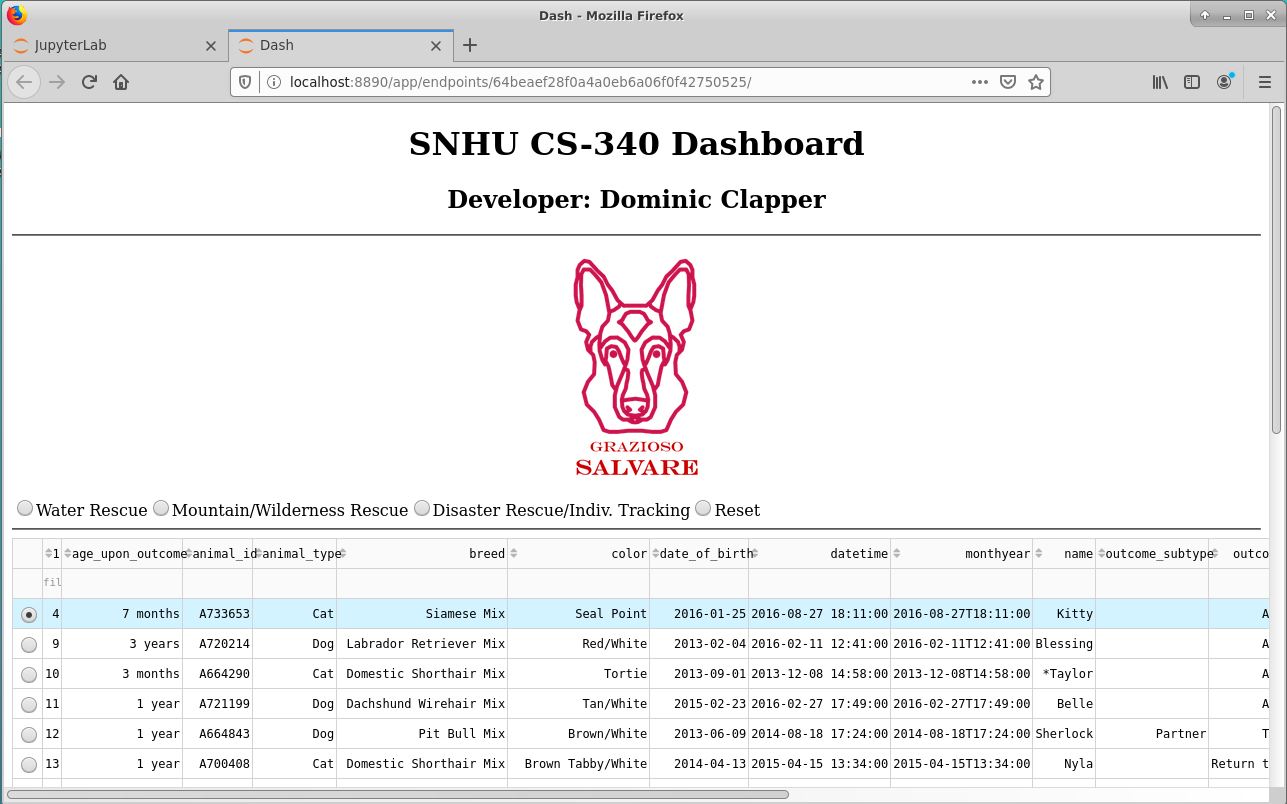


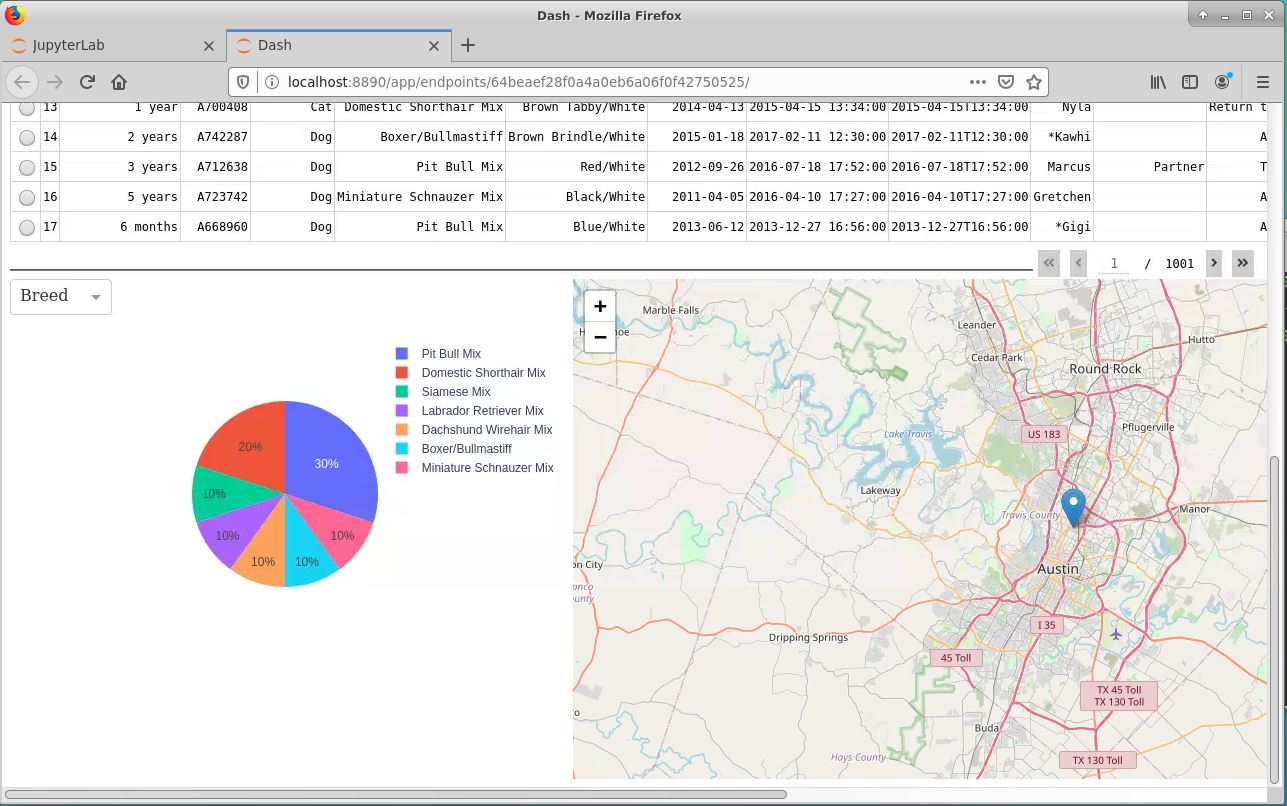


* Now that the IPYNB file and the Database are running, our UI is displayed! We can move the running UI to a new tab or window if necessary for better viewing, but our code works! All that is left to do is test the functionality of the code to ensure the filtering works and that the chart/map works.

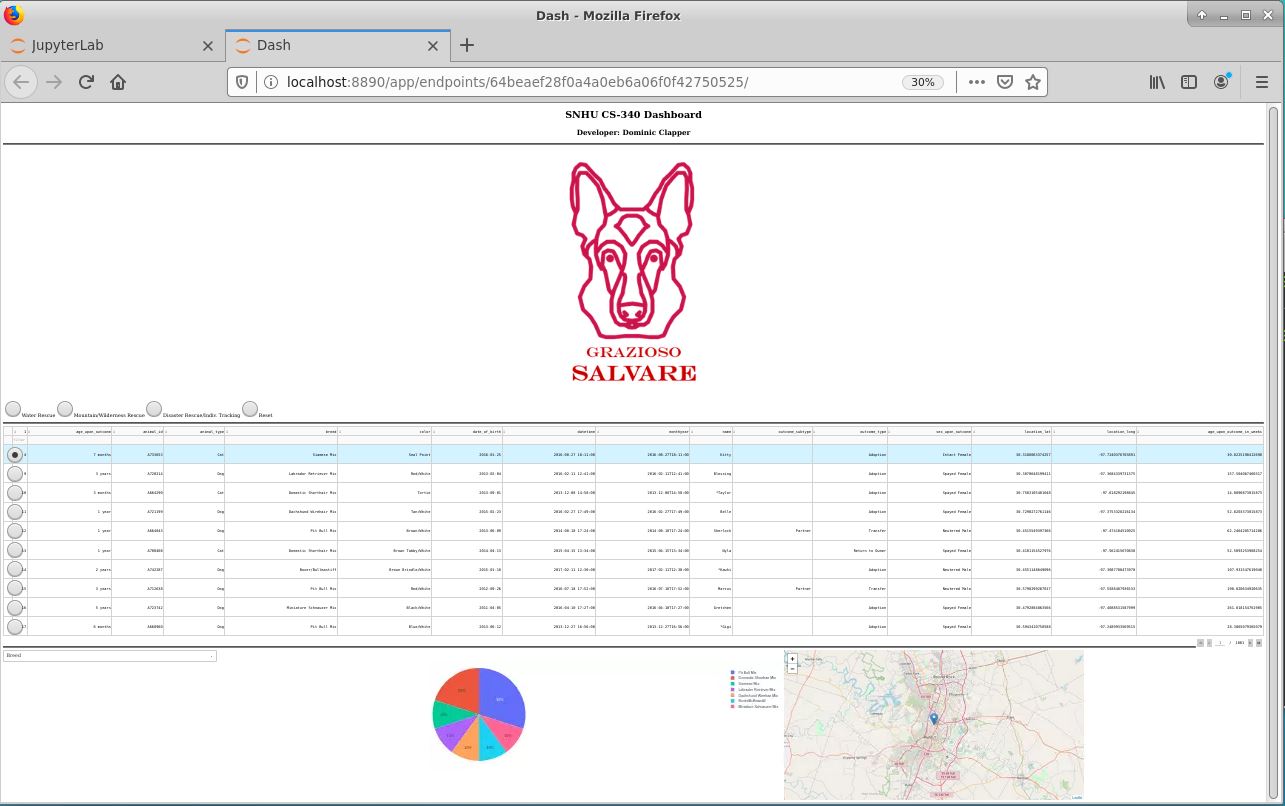
**-Tests**

* As was mentioned in the Code Example portion above, the UI still needs to be tested to ensure that it works.
* To start it off, we will test to see what our starting state of our UI dashboard looks like:



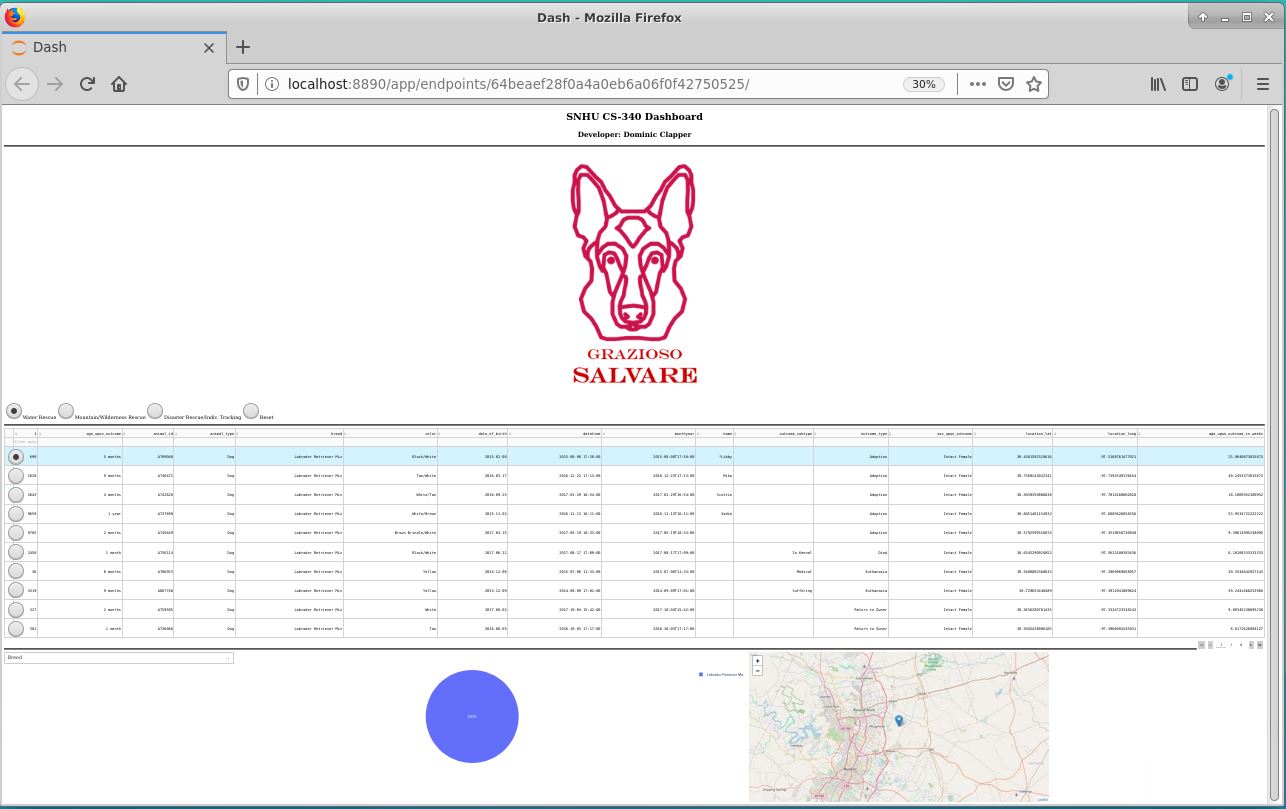
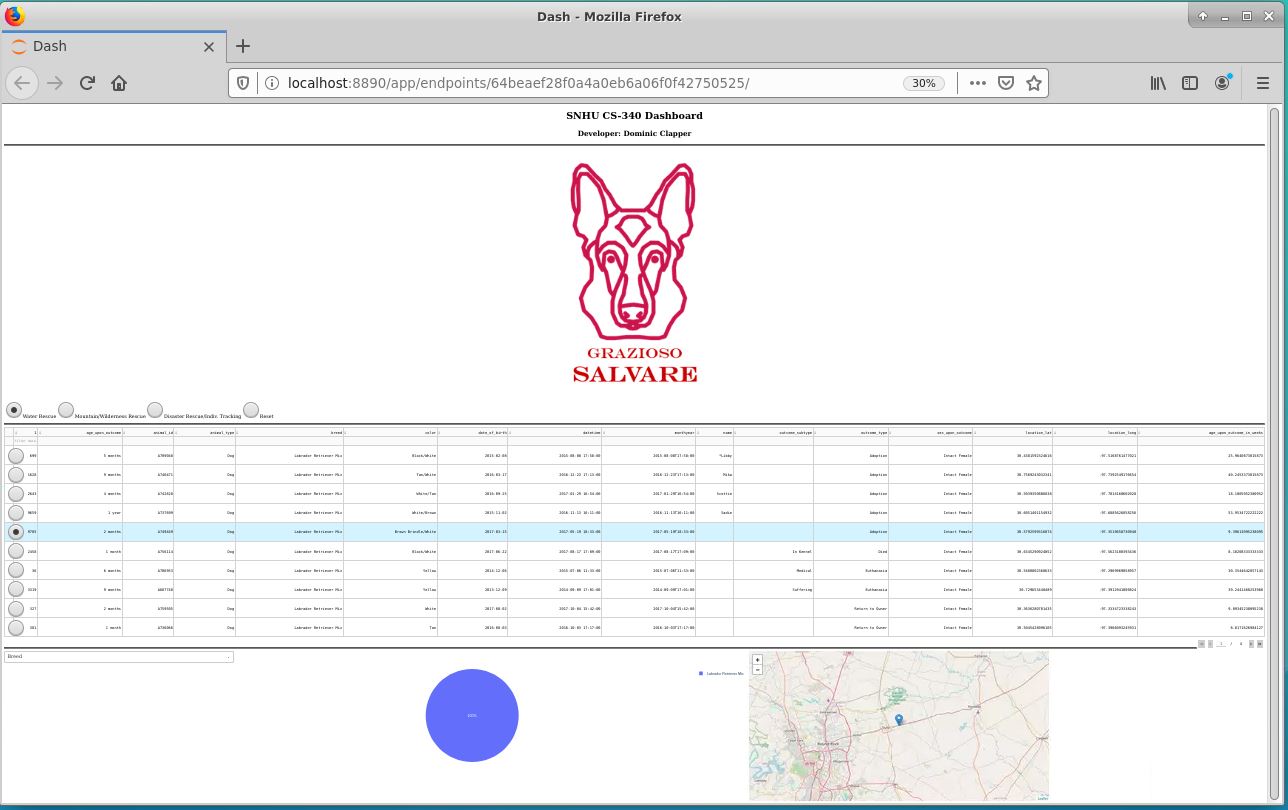
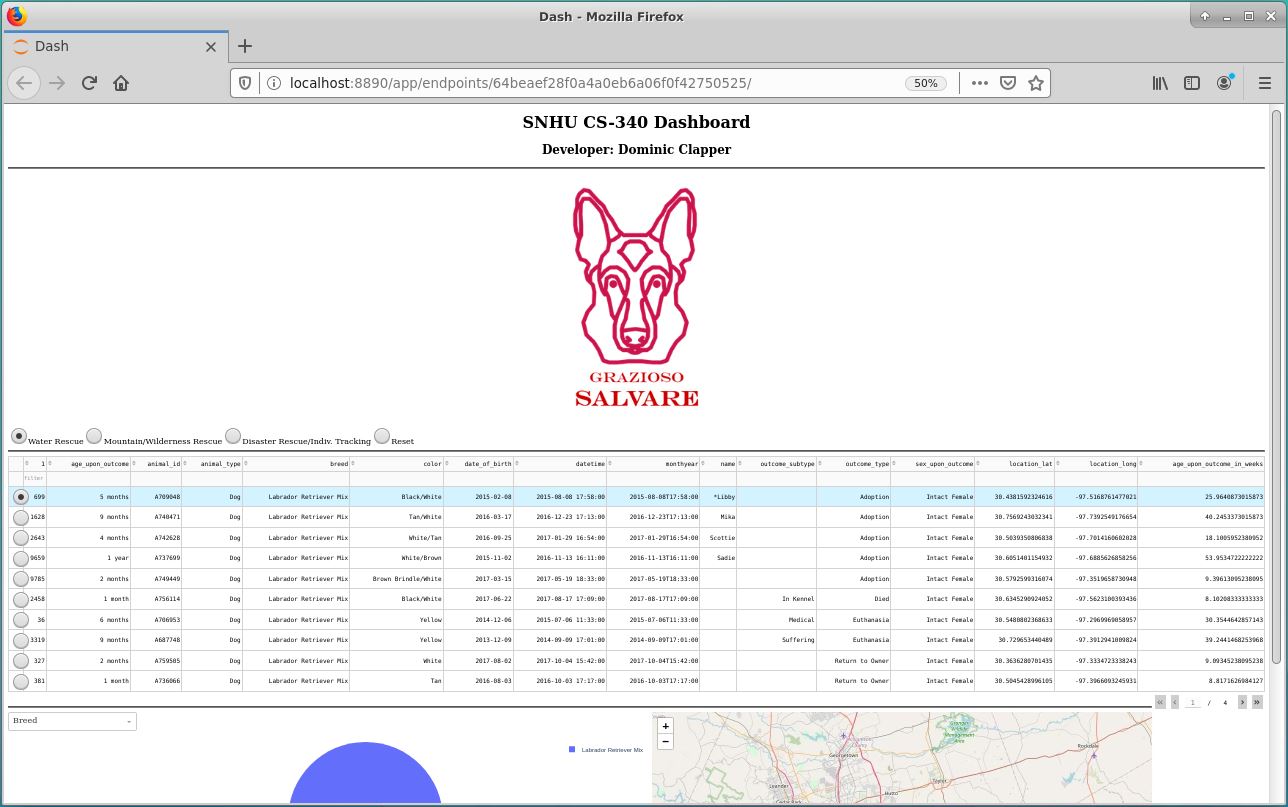


Here is a full screen screenshot of the starting state of the dashboard:

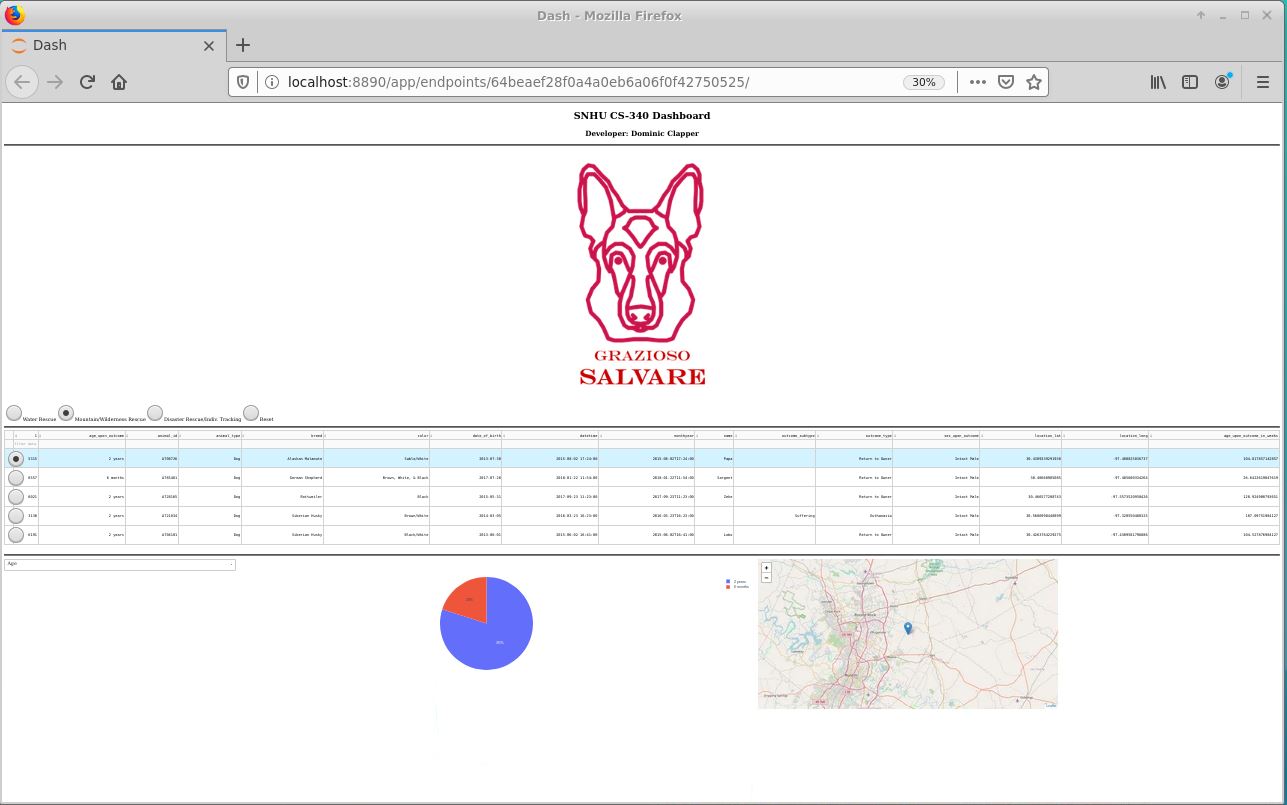
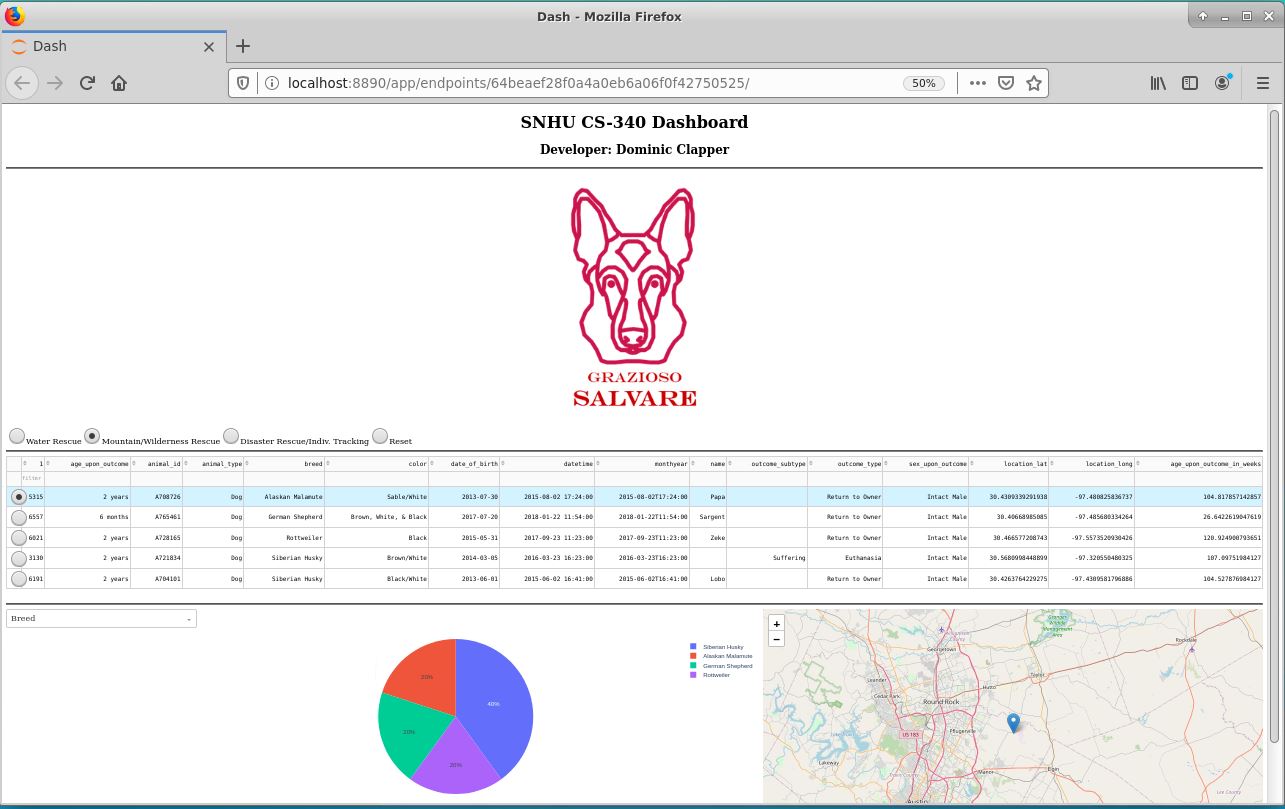


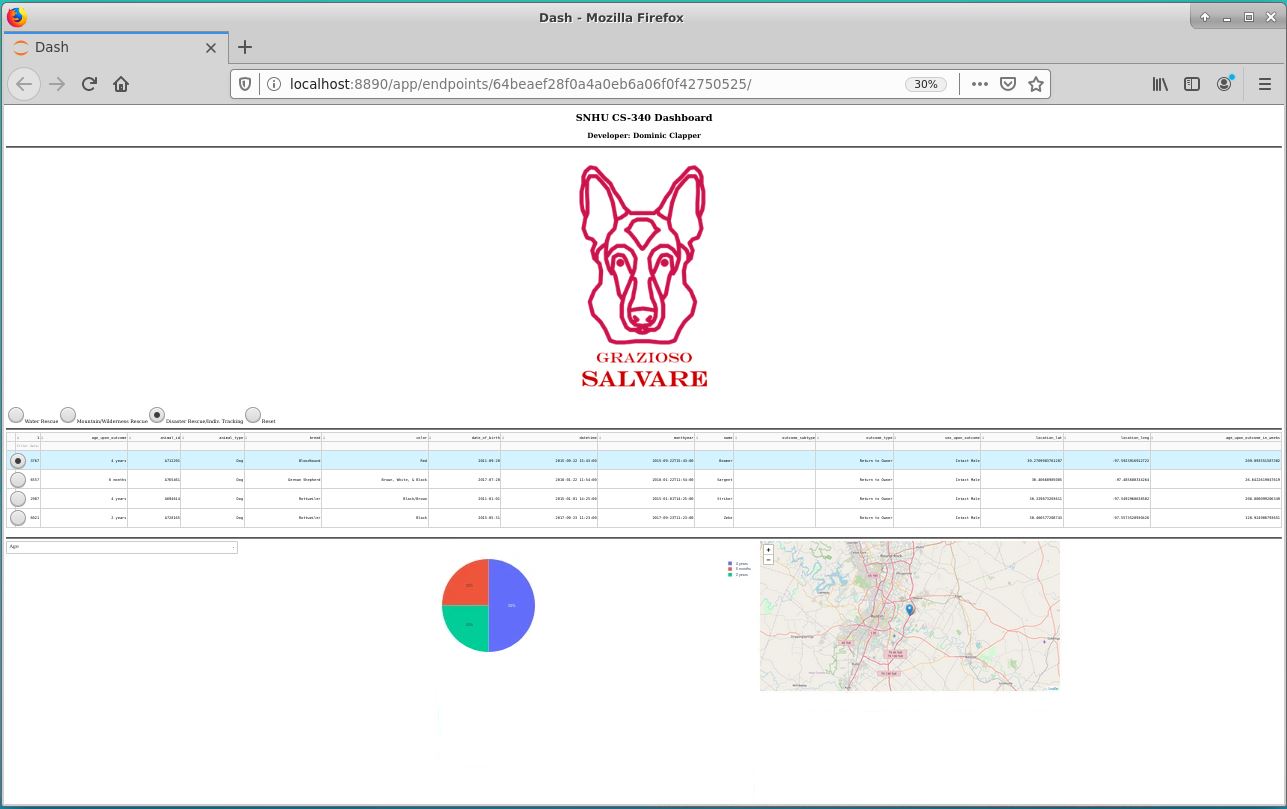
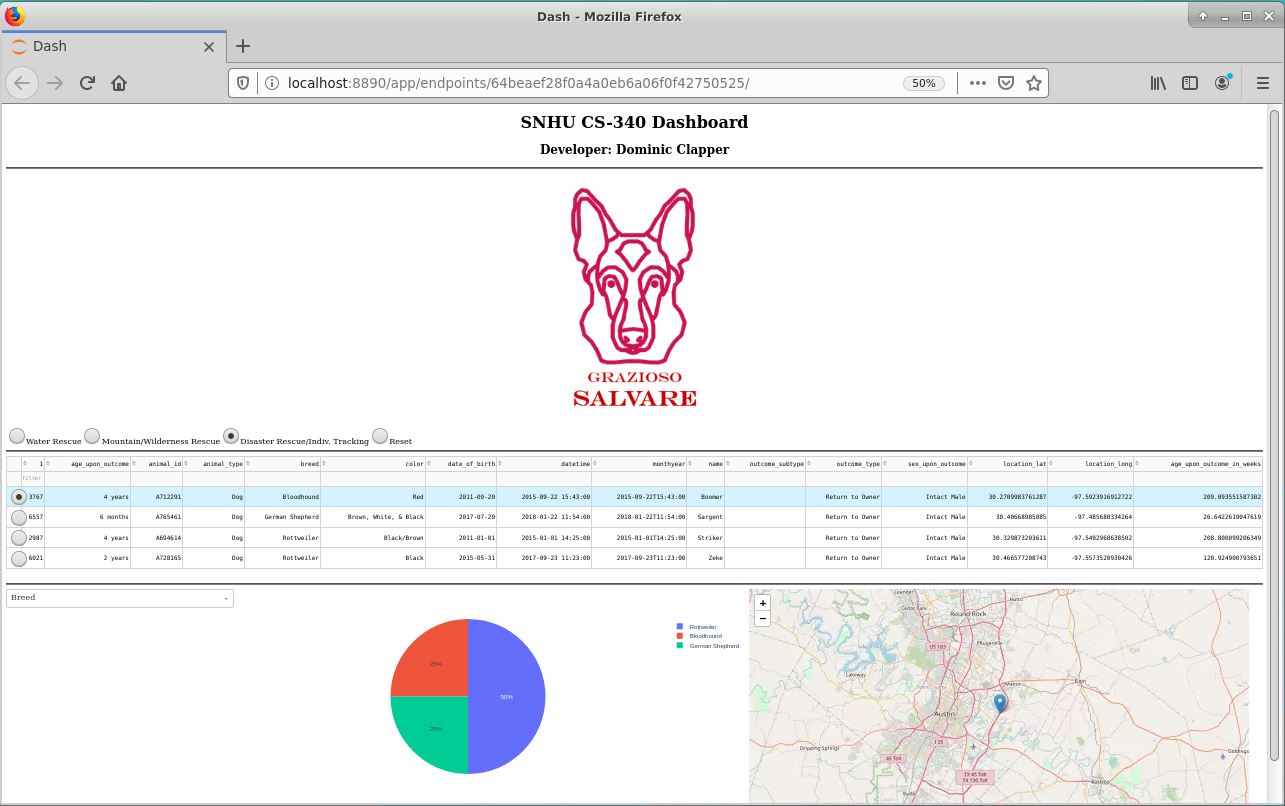
* The next thing we want to test is if the interactive filtering widgets work.

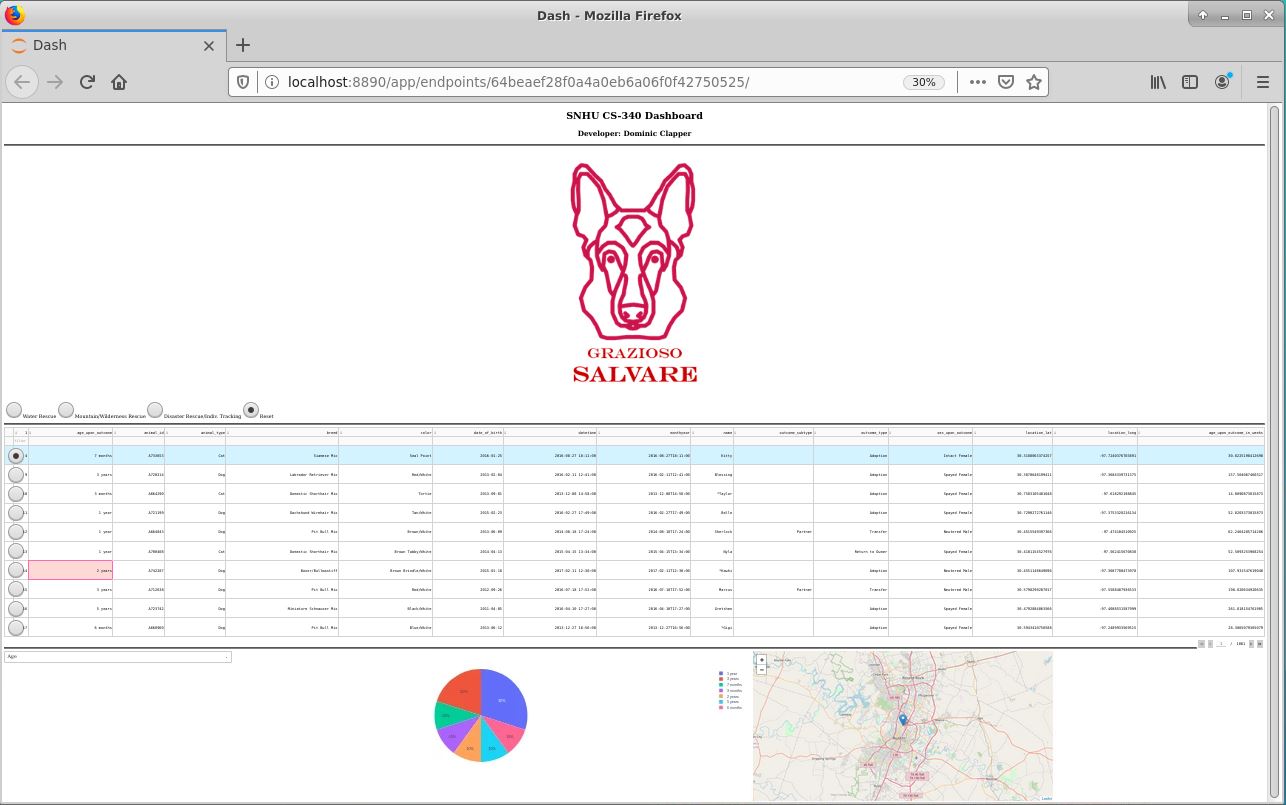
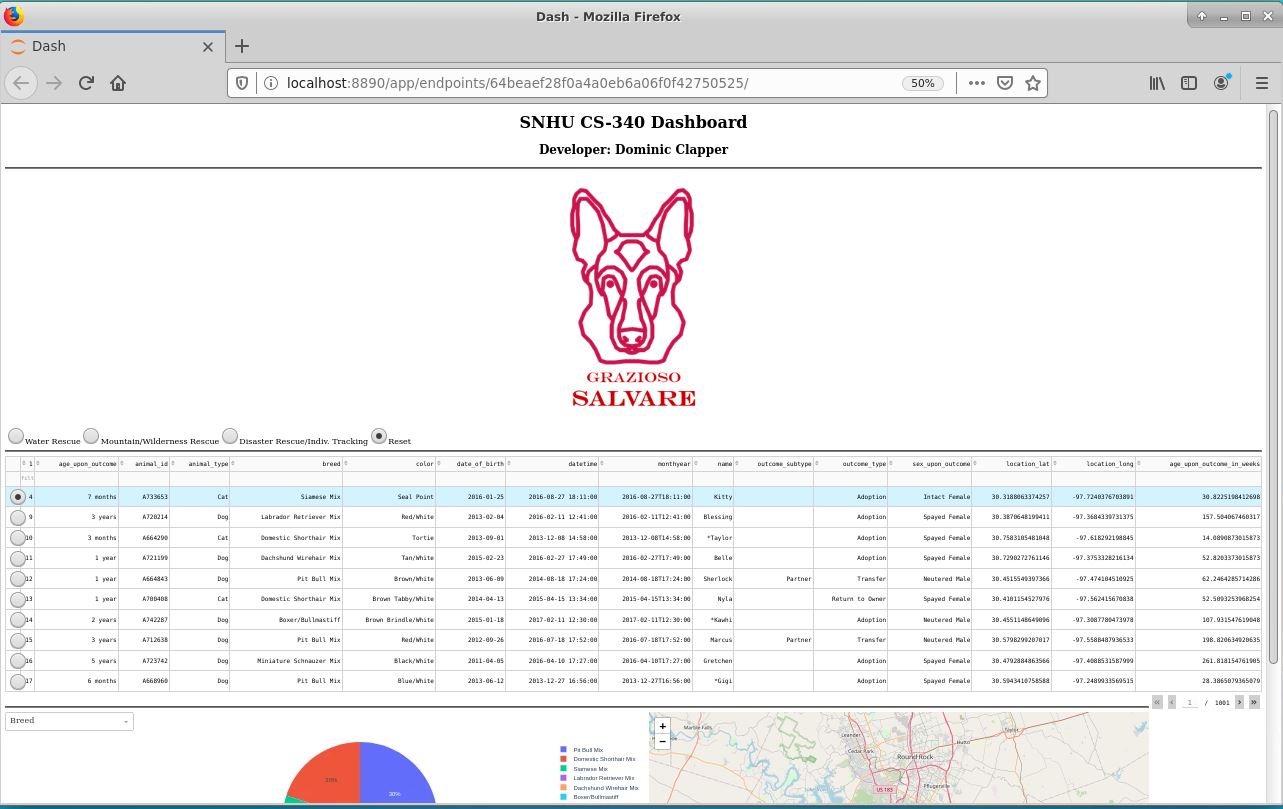
Here is the Water Rescue filter being selected:

It may be a little hard to see, but when a different dog is selected in the table, the geolocation chart changes its position. Also, when the Water Rescue filter was selected, there were only 4 pages worth of dogs listed and most of them were Labrador Retriever Mixes. The UI changed when the Water Rescue filter was selected.

Here is the Mountain/Wilderness filter being selected:

Here is the Disaster Rescue/Indiv. Tracking filter being selected:

Here is the Reset filter being selected:

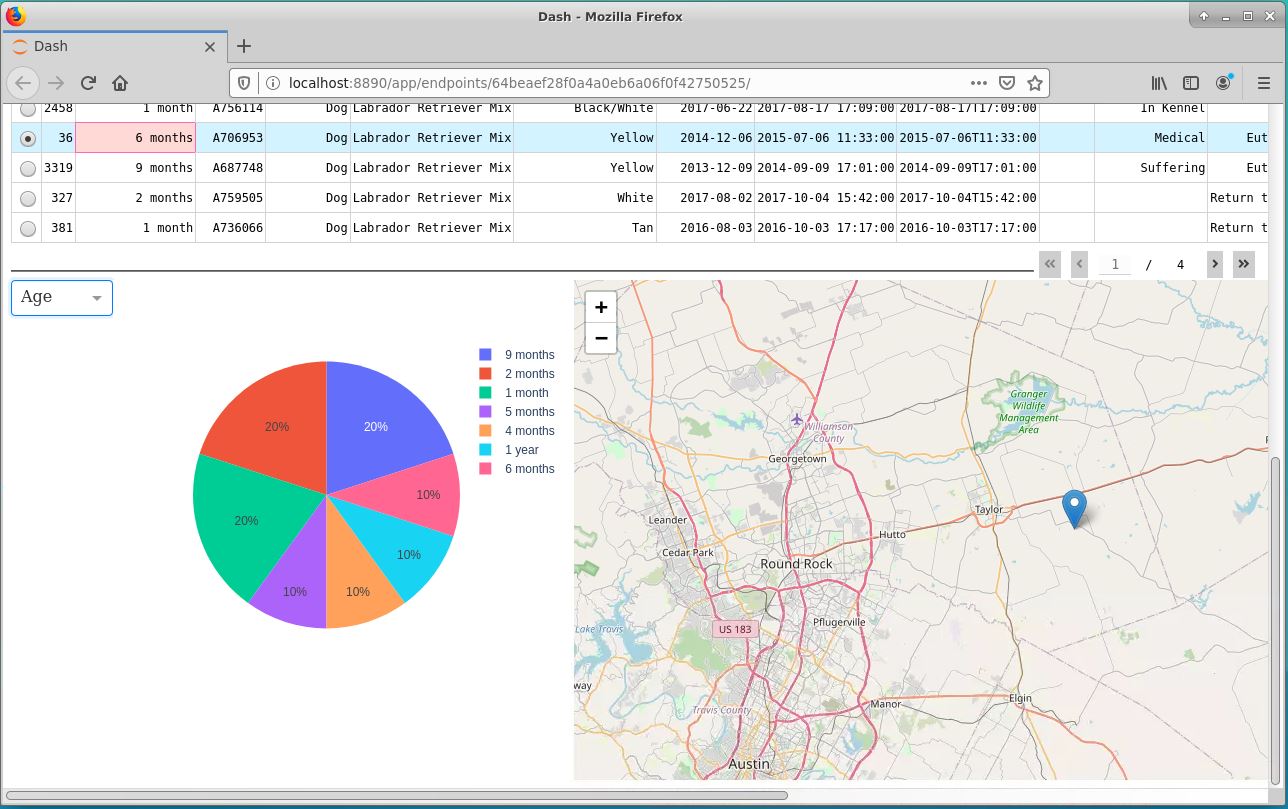


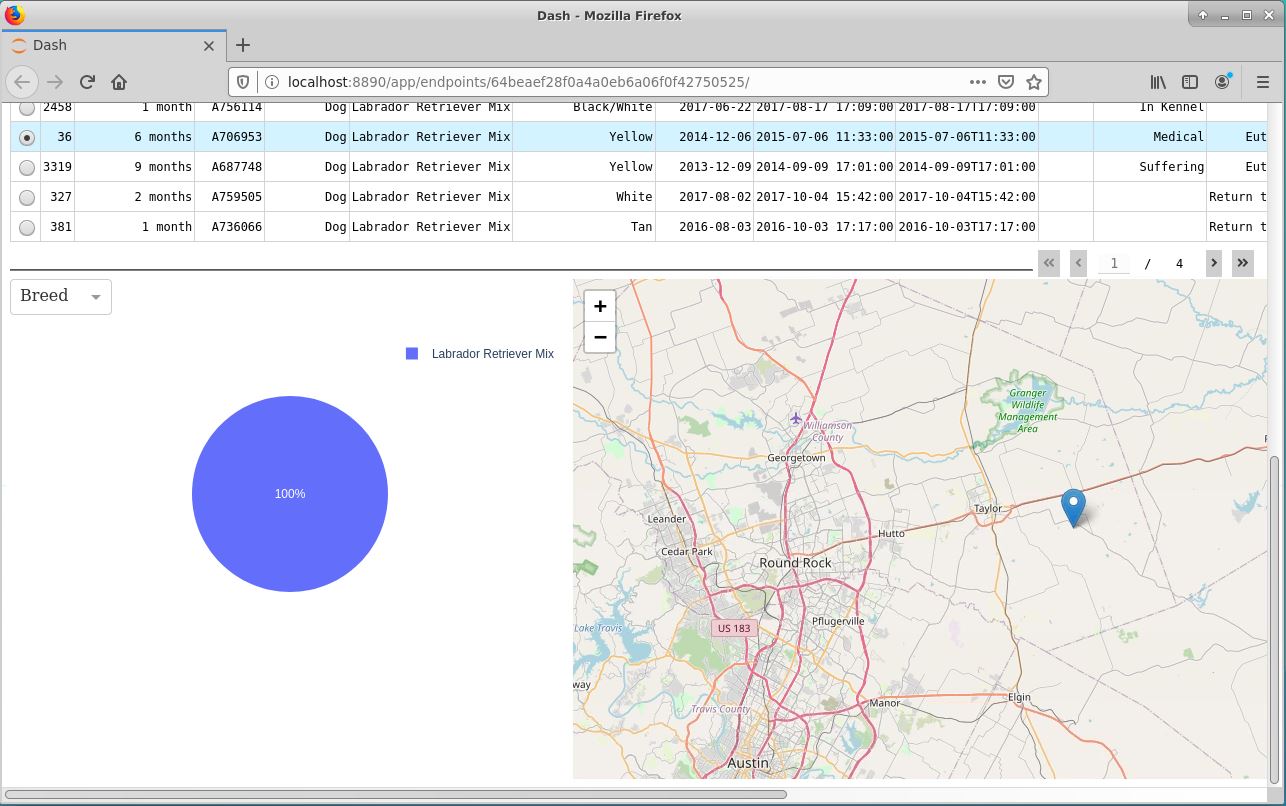
You may notice that in some of the screenshots, the pie chart is a bit different from before. That is because the pie chart also has a filtering option! You can select for the Pie chart to either show the percentages of animals found in terms of age or in terms of breed. This filter option can be accessed by using the dropdown box to the left of the pie chart.

All in all, the above screenshots show that the IPYNB UI works completely! The UI is interactive, and the filtering options work correctly, altering the data table, pie chart, and geolocation map whenever a filter is selected.

**-Screenshots**

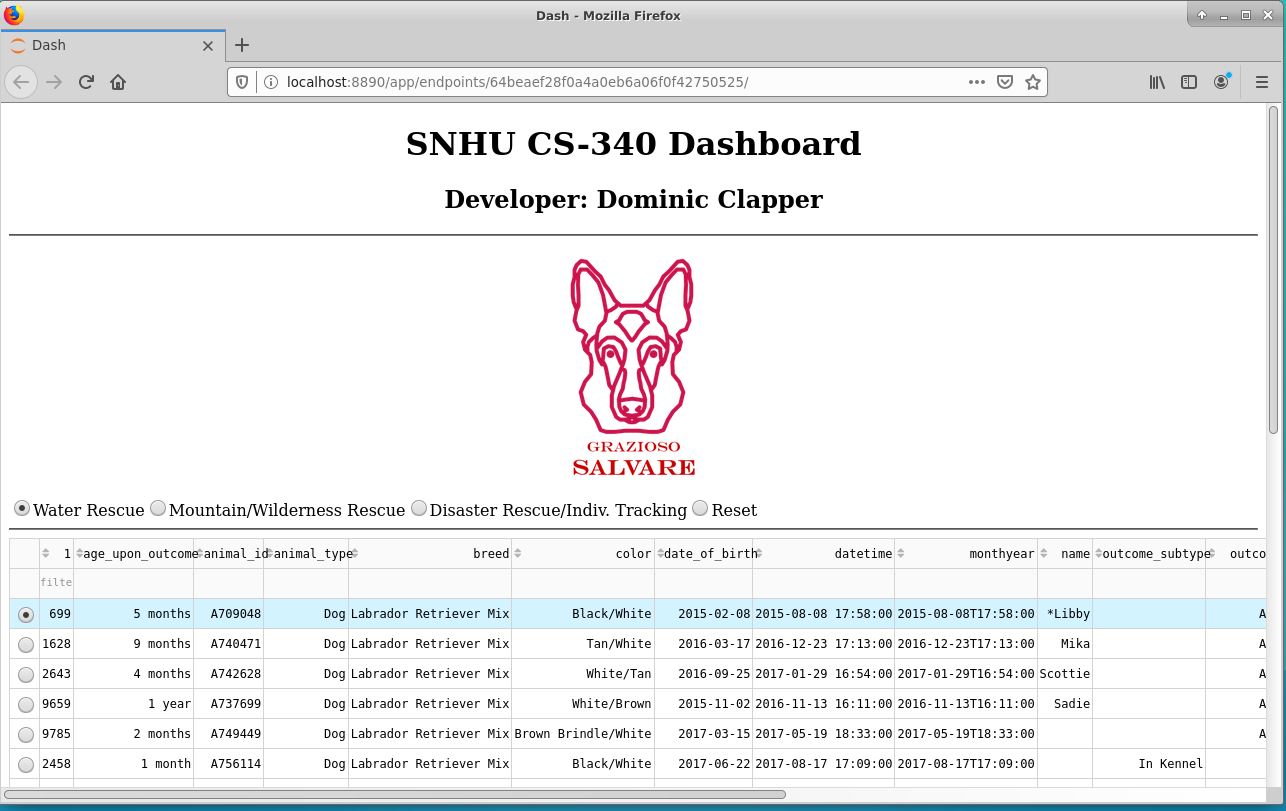
Here are extra screenshots showing the functionality of the UI:





In the above two screenshots, you will notice that the pie chart changes based upon the option selected in the dropdown box filter.

Here is an up close screenshot of the Water filter being selected:



**-Complications/Challenges**

During the development of this project, there were a few roadblocks I faced. One I remember specifically was trying to figure out why my IPYNB file would not run. I kept getting the error that the app = JupyterDash(‘SimpleExample’) line was not working correctly, and it had me confused for so long. Then I went through the code bit by bit and realized that there was a parenthesis missing above the line. Once I placed another parenthesis, the problem went away instantly. This same problem occurred again when I was working in the app.layout area. There is a lot of opening and closing of brackets, braces, and parenthesis, which resulted in me getting stuck for an hour trying to find out where I was missing something. After a while I was able to pinpoint the few locations where I had missed a parenthesis or bracket, and I fixed it with ease. Afterwards, my UI was able to display! Another challenge I faced was trying to get my geolocation map to work correctly. I spent a while on this because there was not much documentation I could find online, so it took a lot of effort. Whenever I would get really frustrated that the code for the map didn’t work, I tried stepping back and taking a small break to let my mind settle down. This helped immensely because it allowed me to refocus and look at the problem from a different viewpoint. In the end, I had to rework all of the code from my module 6 milestone to make it work for the project, but I feel it was worth it because the map works better now.

The only other problem I encountered while working on the project was honestly trying to get good screenshots. It is tough trying to take a screenshot of the entire dashboard without making all the details look super tiny. I tried my best at it, and I think the screenshots I have are decent, but there is definitely room for improvement. My main way of getting slightly better screenshots was to make the window showing the UI smaller. I also opened up the UI in a new window instead of the one beneath the IPYNB code because it allowed me to get a better picture.

**-Contact:**

**Dominic Clapper**