# Dre’ Scheetz

CS 340

Jeff Sanford

02/20/2022

# Search and Rescue Identifier Readme

## About Search and Rescue Identifier

## This full-stack application can identify ideal candidates for search and rescue training from participating animal shelters. By incorporating the provided animal shelter data, the application can filter and display relevant information about each candidate.

## Motivation

## Due to the amount of data, manually searching would be time-consuming. This application allows users to efficiently perform queries and get the needed information promptly while reducing the potential of human error.

## Getting Started

To implement Search\_Identifier, create a database using MongoDB with user Authentication. Inside mongo\_module.py, update the \_\_init\_\_ to your port, database, username, and password—open Search\_Identifier from Jupyter Notebook. MongoDB will also need to be running for the script to function. The following in Search\_Identifier will need to be updated based on your MongoDB database:

*###########################*

*# Data Manipulation / Model*

*###########################*

*username = "aacuser" # Updated username for Mongo*

*password = "user" # Updated password for Mongo*

*shelter = AnimalShelter(username, password)*

*# class read method must support return of cursor object*

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

User authentication is essential to the script's operation; CRUD operations cannot be performed without it. Here is an example of creating additional users per <https://docs.mongodb.com/manual/tutorial/create-users/> :

use test

*db.createUser(*

*{*

*user: "myTester",*

*pwd:*

*passwordPrompt(), // or cleartext password*

*roles: [ { role: "readWrite", db: "test" },*

*{ role: "read", db: "reporting" } ]*

*}*

*)*

Note that in this example, this user would be unable to use the create method of mongo\_module for the database reporting. This can be changed by using the grantRolesToUser() function in MongoDB to change the role to readWrite. For switching users, exit MongoDB then stop the database using /usr/local/bin/mongod\_ctl stop. Proceed to start the database with /usr/local/bin/mongod\_ctl start then use mongo --authenticationDatabase "DB\_NAME" -u "YOUR\_USERNAME" -p and enter your password.

After this setup, Search\_Identifier should be ready for use. Run the Jupyter Notebook cell. Functionality includes pagination for selecting the number of rows displayed in the data table, RadioItems filtering for performing a quick query of MongoDB based on hard-coded parameters, and graphs for visualizing the data.

MongoDB is the base for this application. MongoDB provides a NoSQL document database that performs efficient CRUD operations. Some of the benefits over other more traditional relational databases are the simple inclusion of data without a complex schema, simple querying with robust parameter options, and scaling. PyMongo, Bson.json, and Dash libraries are used in this project. PyMongo is an excellent tool for interacting with MongoDB and is considered a requirement for this project. The Bson.json allows for using the Mongo\_Module read function for proper formatting of the returned query. Dash provides for the user-friendly interface and readability of the data. It also offers extensive interactivity of this web application with robust analytic capabilities.

## Installation

Any text editor should suffice for editing the mongo\_module although PyCharm is recommended. It’s a very popular IDE.

For PyCharm installation: <https://www.jetbrains.com/pycharm/download/#section=windows>

MongoDB can be installed via the Installer.

* Installation instructions for MongoDB: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>
* Download link: <https://www.mongodb.com/try/download/community?tck=docs_server>

Jupyter Notebook will also need to be installed.

* Installation is done with either conda or pip
* Installation instructions for Jupyter: <https://jupyter.org/install>

After downloading the required tools and following the getting started section you can upload the Search\_Identifier.ipynb and mongo\_module.py from inside Jupyter Notebook.

## Usage

### Code Example

Mongo\_module.py:

*def \_\_init\_\_(self, username, password):*

*# Initializing the MongoClient.*

*self.client = MongoClient('mongodb://%s:%s@localhost:39009/AAC' % (username, password))*

*self.database = self.client['AAC']["animals"]*

*def read(self, query: dict):*

*if query is not None:*

*# Assigns results of mongoDB query to result\_cursor*

*result\_cursor = self.database.find(query, {"\_id": False, "monthyear": False,})*

*if result\_cursor.count() == 0:*

*raise Exception("No results for query")*

*else:*

*return result\_cursor*

*else:*

*raise Exception("Nothing to search, query parameter empty")*

SearchIdentifier.ipynb:

*dcc.Dropdown(*

*id='select\_page\_size',*

*options=[*

*{'label': '5', 'value': 5},*

*{'label': '10', 'value': 10},*

*{'label': '15', 'value': 15},*

*{'label': '20', 'value': 20},*

*],*

*# Default value for results per page.*

*value=10,*

*style=dict(width='40%', verticalAlign="middle")*

*),*

### *dcc.RadioItems(*

### *id='filter-type',*

### *options=[*

### *{'label': 'Water Rescue', 'value': 'water\_filter'},*

### *{'label': 'Mountain Rescue', 'value': 'mountain\_filter'},*

### *{'label': 'Disaster Rescue', 'value': 'disaster\_filter'},*

### *{'label': 'Reset', 'value': 'no\_filter'},*

### *],*

### *value='no\_filter',*

### *labelStyle={'display': 'inline-block'}*

### *)*

*def update\_dashboard(filter\_type): # Filtering function for interactive data table.*

*if filter\_type == 'water\_filter':*

*# Mongo\_Module CRUD method using the given filter parameters.*

*df = pd.DataFrame(list(shelter.read({'$and': [{'$or': [*

*# $regex is used due to varied inputs for mixed breeds*

*{'breed': {'$regex': 'Labrador'}},*

*{'breed': {'$regex': 'Newfoundland'}},*

*{'breed': 'Chesapeake Bay Retriever'}]*

*},*

*{'sex\_upon\_outcome': 'Intact Female'},*

*# $gte and $lte are used for the age range as greather than or*

*# equal to and less than or equal to*

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

*]*

*})))*

### Tests

Testing the crud module can be done with doctest or unittest directly in the notebook from additional cells of Jupyter Notebook. Leaving the curly braces empty or using an invalid input will output exceptions.

Please see the documentation for performing tests of the dash components <https://dash.plotly.com/testing>. User acceptance testing was the primary method during the development of this application.

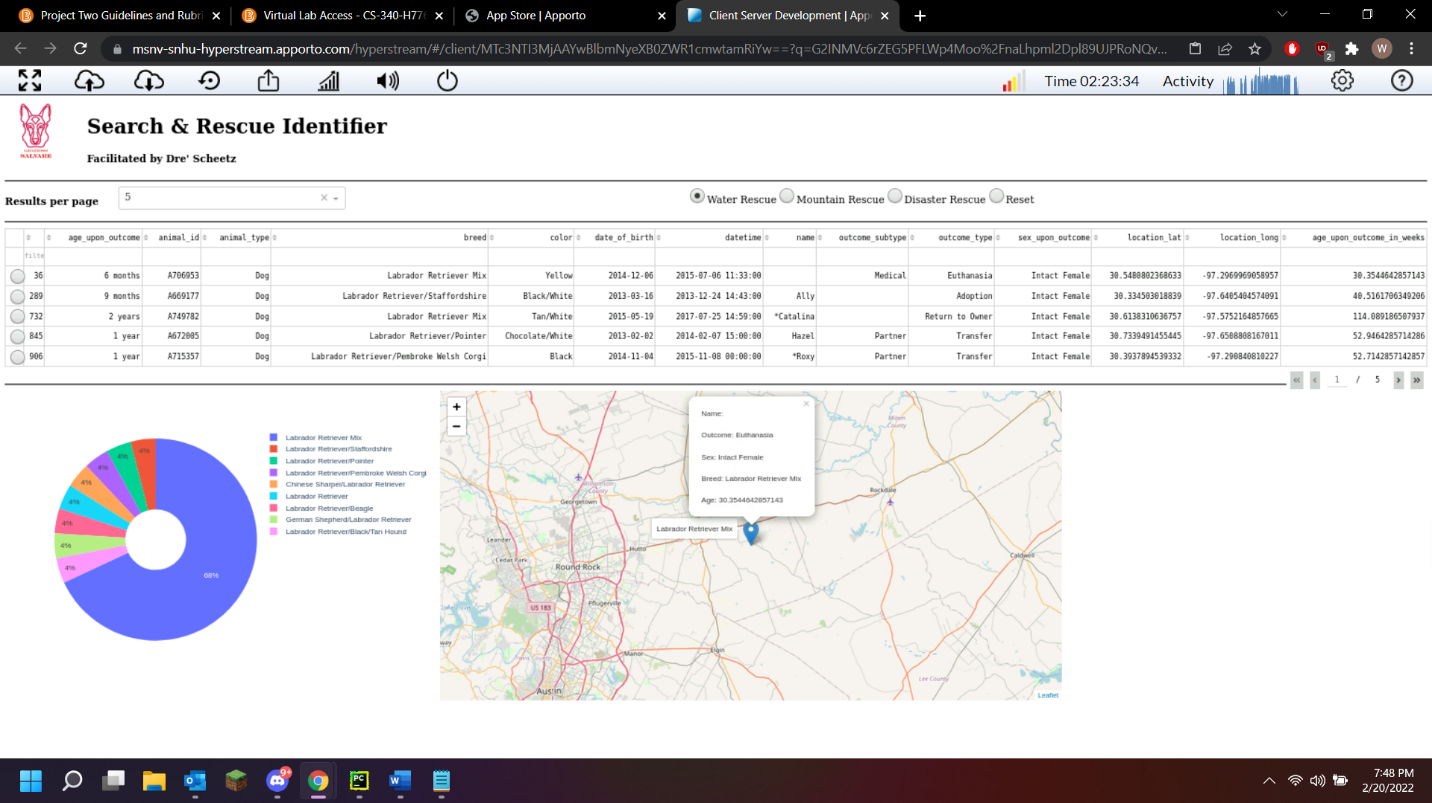
### Screenshots:

Screenshot 1: Starting state of application. By default, the app loads to a data table pagination of 10 and on the reset (no filter) option. Also, hovering over the map marker.

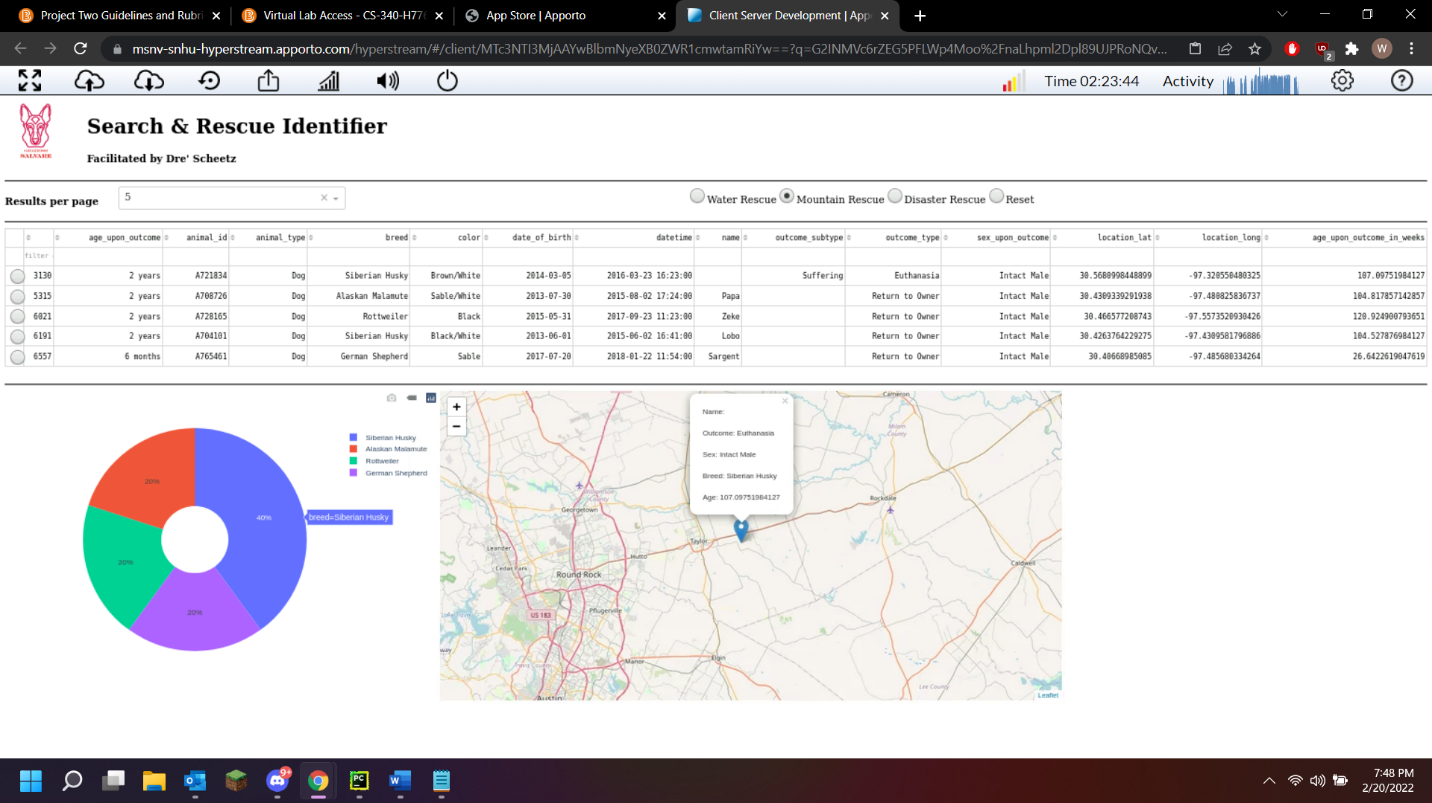
Graphical user interface, application

Description automatically generated

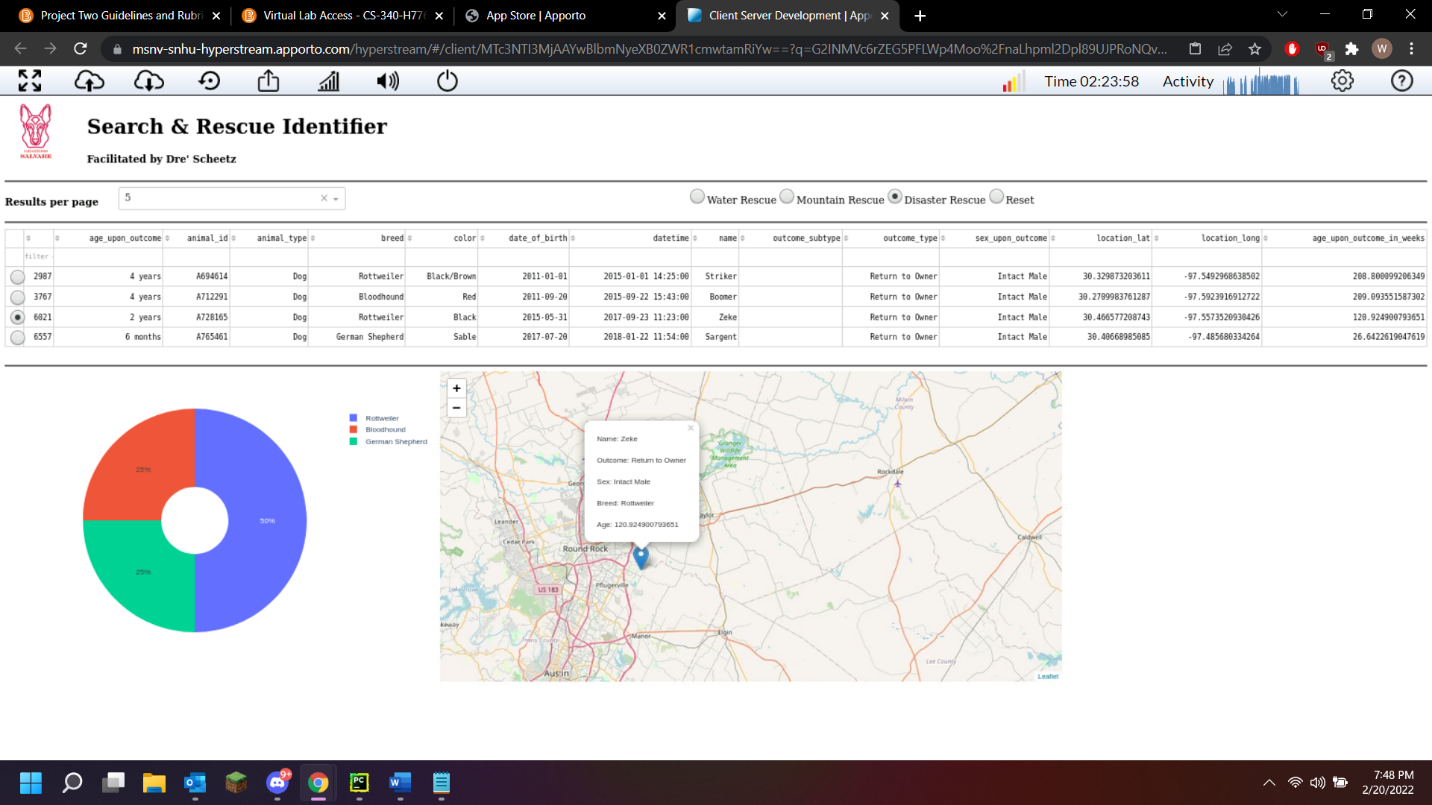
Screenshot 2: Water Rescue option selected, results per page set to five. The map marker has now been clicked on to display relevant information. The map also centers on the map marker.



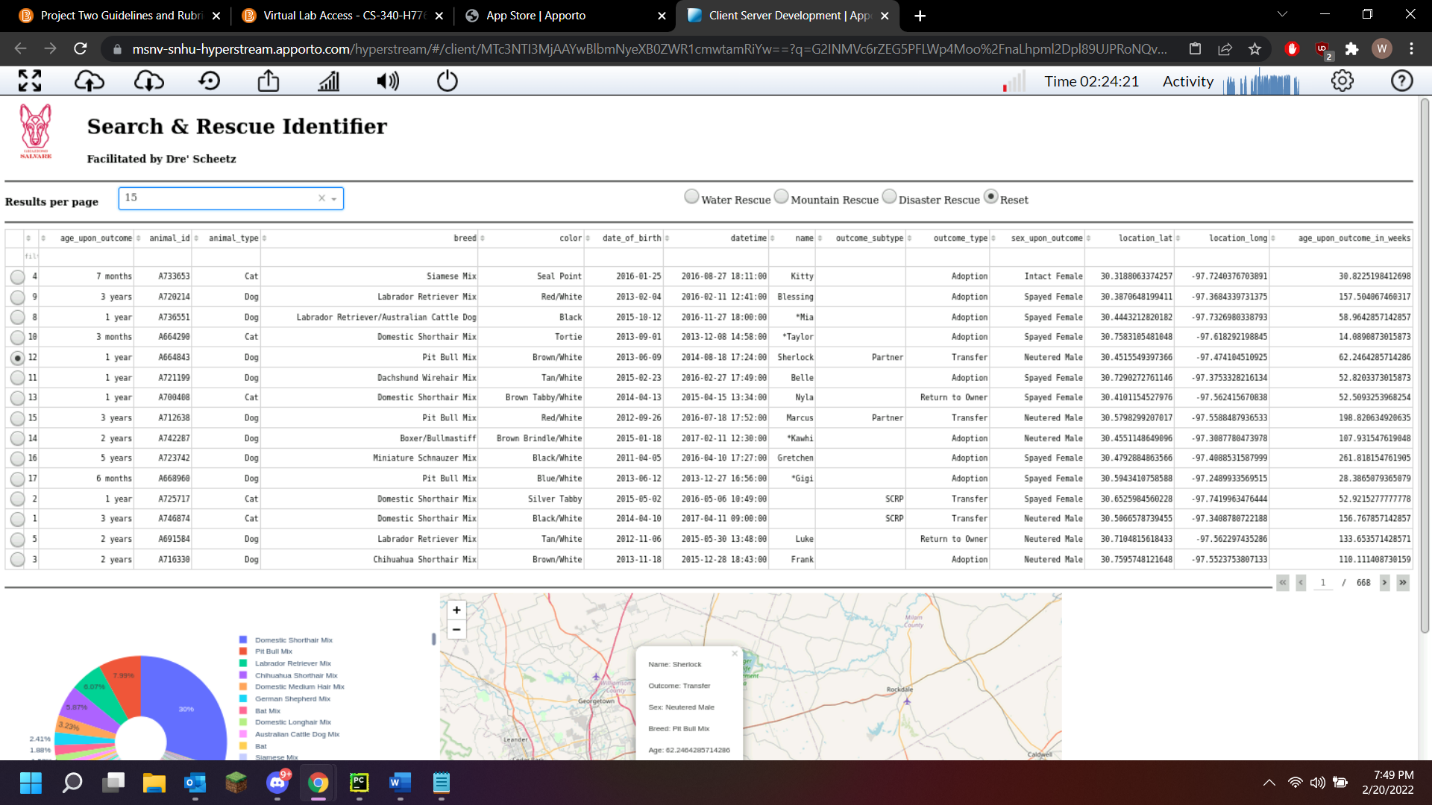
Screenshot 3: Mountain Rescue selected, hovering cursor over pie chart.



Screenshot 4: Disaster Rescue selected along with an animal from the data table to show that a single row may be selected which updates the map marker.



Screenshot 5: Reset option selected from RadioItems showing the unfiltered data table. Pagination has also been changed to 15.



## Roadmap/Features

* Grouping of less than a set percentage to “Other” for the pie chart.
* Map Markers based on the page\_size of the data table.
* Reformatting of the CSV file used so the most relevant cells appear left to right.
* Better sizing of components for operation when not in full screen.

## Challenges

## The most significant challenge of the web application was the graph and map used. Many attempts were made to include a for loop based on page\_size of the data table, so each row displayed a map marker. Unfortunately, there is a lack of documentation for this. During development, it was determined that grouping breeds less than a percentage was out of scope for this project. The querying of MongoDB also became problematic when attempting to incorporate $regex into the $in operator. $or was used instead due to time limitations. While this would work in MongoDB, I could not find any examples of this implementation with the provided read method in Dash Jupyter. The only other challenges are the ones found in the CRUD Readme that were still prevalent.

## Contact

Dre’ Scheetz

dre.scheetz@snhu.edu