Southern New Hampshire University

CS-340: Assignment 7-2: Writeup to Project 2

Prepared on: October 21, 2025
Prepared for: Prof. Jeff H. Sanford
Prepared by: Alexander Ahmann

Contents

1 Background

This writeup documents my experiences assuming the role of the lead developer for the fictious *Global Rain* software shoppe.[1] In particular, the equally fictious *Grazioso Salvare* is interested in our services. They are an organisation that “identifies dogs that are good candidates for search-and-rescue training [...] [who can] help rescue humans or other animals, often in life-threatening conditions.”[2]

As described in the project guidelines (CS-340, n.d.-a, Scenario) and a given specifications document (CS-340, n.d.-b), a web application is the proposed solution to identify animals that have the potential to go on rescue missions. Such a web application must communicate with a server hosting an instance of some kind of database, and render its data into tables, graphics, and charts that make it easier for end users to make sense of the raw datasets, and to quickly and accurately make the best decisions for the selection of candidate rescue animals. This writeup serves the additional purpose of documenting the solution to the specific problems and constraints outlined in the project guidelines.

1.1 Task 1: Formulation of a Problem Statement

> “Before developing the Python code for the dashboard, be sure to review the Dashboard Specifications Document provided by the UI/UX developer at Global Rain.[3] In addition to the widgets, you have been asked to include the **Grazioso Salvare logo and a unique identifier** containing your name somewhere on the dashboard. A high-resolution copy of the logo is included in the `code_files` directory in Codio.”—CS-340 (n.d.-a, Prompt: §1)

When formulating a problem, I do not intend to focus too much on trivial stuff, like “where the unique identifier ought to be placed.” It would be better to discuss the more important stuff— that being the solutions that

[1] Of course, this is all for a programming course. The real objective here is for this student to demonstrate mastery of “[the application] of database systems concepts and principles in the development of a client/server application” and “[the development] client-side code that interfaces with databases.” (CS-340, n.d.-a, Compentencies)

[2] Quoted verbatim from CS-340 (n.d.-a, Scenario).

[3] This document is located in the Supporting Materials section and will provide you with examples of the different dashboard widgets you will create (CS-340, n.d.-b)

address creating a web application by which to interface with a database, and display information in the form of interactive tables and charts. The database management system of choice that I have decided on is *MongoDB*: a non-relational database management that stores information as "documents" in a collection, in which data is modelled through transactions involving accessing or modifying values stored in keys. The web application must deploy a *model-view-controller* approach. Specifically, I have decided to go with Python's Dash framework for implementing the front-end component of the web application.

As described in the specifications document (CS-340, n.d.-b, p. 1), the important stuff includes options by which to filter results based on a set of criteria, an interactive data table, a geolocation chart, and a second chart of the student's choice.[4] Later sections, and subsections, elaborate further on the solutions to the specific problems involved in the invention of the web application by which this problem is solved.

2 Review of Previous Work

2.1 The Setup and Configuration of a Non-Relational MongoDB Instance

This project is something of a "full-stack" web application: with a backend database, a custom library by which to perform CRUD operations against the database, and a front-end web application by which to present the information to an end-user. The backend consists of a non-relational database called *MongoDB*, which stores information in the form of "JSON-like" documents which stores data in the form of a `HashMap`-like data structure consisting of `keys` and their respective `values`.[5]

For this part of the project, I have setup the MongoDB instance to include a custom database called `aac`, and used the `mongoimport` tool to import the Animal Services (2016) dataset— formatted in the *comma seperated values* (CSV) format, which is like a tabular dataset.[6] The resulting document collection is called `animals`— with the MongoDB database and collection that is the subject of this project is known as `aac.animals`.

[4]In this section of the formulation of a problem, I am, to some degree, vague in my descriptions and what constraints that I have to work with. The "project guidelines" gives mostly detailed instructions on what to do, giving me the liberty to not have to worry too much about specificity.

[5]In MongoDB parlance, these are called *BSON documents,* or "Binary JSON" documents.

[6]Ahmann (2025a, §3.1).

Next, for the sake of good ergonomics and cybersecurity, I implemented a *role-based access control* (RoBAC) policy on the MongoDB instance that limits privileges based on what "roles" are assigned to end-users or software tools that use said MongoDB instance.[7] I created a user called `aacuser`, and configured this user to only have *read/write* access to the `aac` database.[8] With this infrastructure up in place, I can now proceed with inventing a web application to visualise the documents in the `aac.animals` collection.

2.2 The Invention and Refinement of a Python-Based, Client-Side MongoDB Interface

Software engineers typically find themselves in situations where they need their applications to run the same procedure multiple times, with some variation in how they are parameterised. It would be inefficient to rewrite these procedures, so software engineers typically write functions, classes of functions, and entire libraries and modules, in order to avoid this kind of redundancy. Indeed, in the development of a web application to visualise MongoDB documents, I found myself inventing a simple "Python CRUD module," which allows for myself and other developers to reuse Python functions that perform the usual *create, read, update,* and *delete* operations associated with databases.

This module is implemented and saved into a file called `CRUD_Python_Module.py`, with the main functionality stored in a class called `AnimalShelter()`. This class inherits functionality from `pymongo` class, and extends its functionality in accordance with the project requirements. Figure 1 depicts something of a "pseudo UML diagram" that documents the module's `AnimalShelter()` class and its respective functions.

[7] For a comprehensive treatment of role-based access controls, see Sandhu (1997); for a smaller introduction, see Lindemulder & Kosinski (n.d.).

[8] Ahmann (2025a, §3.2).

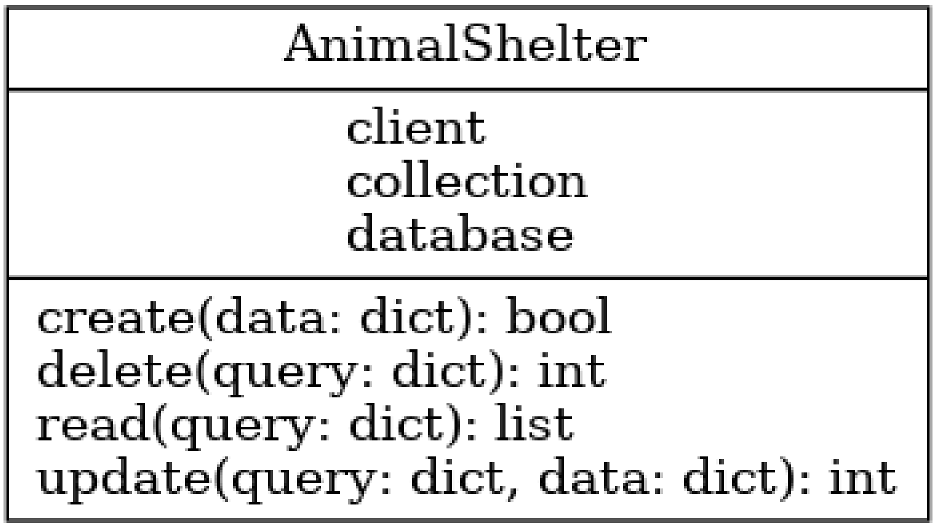


Figure 1: UML Diagram of the Python CRUD Module.

This UML diagram lists the functions: `create`, `read`, `update`, `delete`—which performs their respective CRUD functions. The CRUD module's main functionality consists of the CRUD's class, which is initialized by the `__init__` constructor that takes in a username and password to authenticate to, and the hostname of the MongoDB instance to connect to.

Figure 2 depicts a flowchart diagram of the constructor's logic. A verbal explaination of the `__init__` constructor's procedure is to take in a username `USER`, a password `PASS` and the hostname of the MongoDB instance `HOST`. They are all expected to be strings, and if they are not, an exception is raised. Finally, the `MongoClient` is used to connect to the MongoDB instance with the given credentials, and custom functions are implemented to perform CRUD operations.

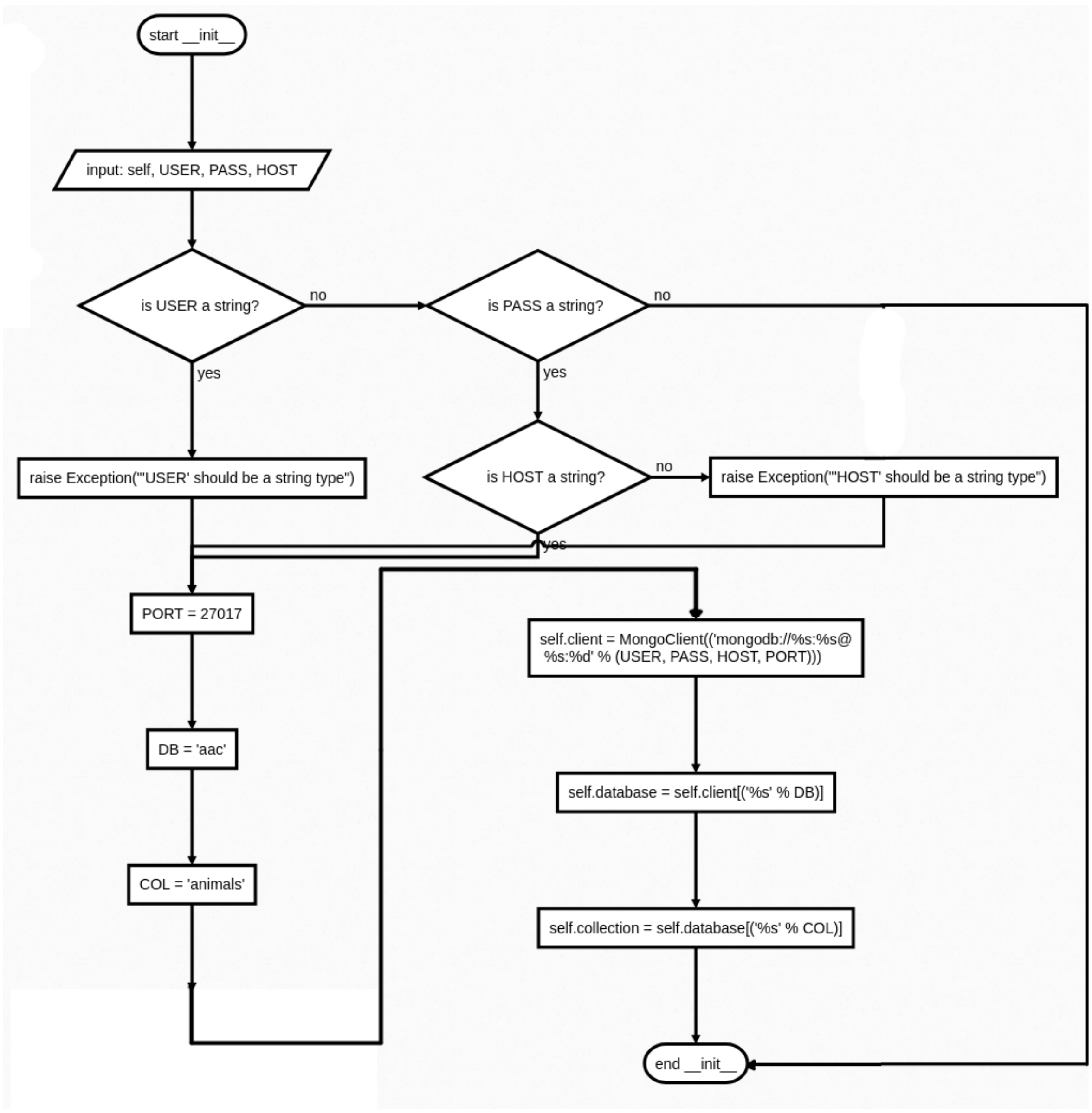
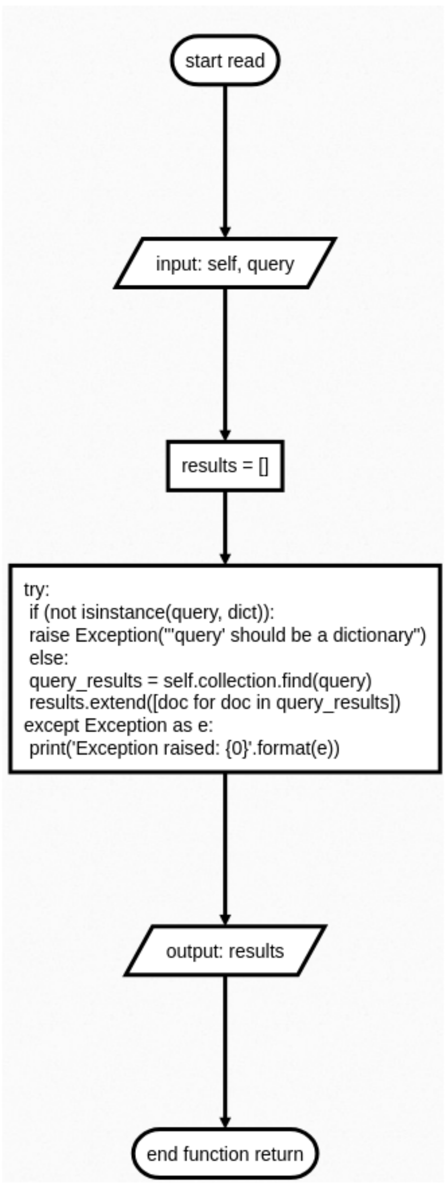

start __init__
input: self, USER, PASS, HOST
is USER a string?
no
yes
is PASS a string?
no
yes
raise Exception("'USER' should be a string type")
is HOST a string?
no
raise Exception("'HOST' should be a string type")
yes
PORT = 27017
self.client = MongoClient(('mongodb://%s:%s@%s:%d' % (USER, PASS, HOST, PORT)))
DB = 'aac'
self.database = self.client[('%s' % DB)]
COL = 'animals'
self.collection = self.database[('%s' % COL)]
end __init__

Figure 2: Logical abstraction of `AnimalShelter()`'s constructor.



```
try:
 if (not isinstance(query, dict)):
 raise Exception("'query' should be a dictionary")
 else:
 query_results = self.collection.find(query)
 results.extend([doc for doc in query_results])
except Exception as e:
 print('Exception raised: {0}'.format(e))
```

Figure 3: Logical abstraction of `AnimalShelter()`'s `read()` function.

There are four functions in the `AnimalShelter()` class, but I will only focus on the `read` function because it is the most relevant one to data modelling, which is the "focal point" of this project.[9] Figure 3 depicts a flowchart for `read`'s procedure. Specifically, it takes in one input: a dictionary that

[9]See Ahmann (2025a) for a more "in-depth" discussion of the Python CRUD Module's code.

acts as a query. The `query` *must* be a dictionary, and will raise an exception if it is not. The function then declares a list called `results`, which is intended to store the returned documents, and uses exception handling to carry out a `find` database transaction. In the case of an exception or lack of results, an empty list is returned, and if no such exception or lack of results are raised, then the `read` function returns the documents that meet the criteria specified in the `query`.

From this module, a front-end web application can query an external MongoDB instance, and can perform data analysis and visualization.

2.3 Tasks 2 & 5: The Development of an Interactive Tabular Dataset and Geolocation Chart

> "Two necessary components for the Dash web application are needed: an interactive tabular component, and a geolocation chart by which to plot locations based on which row is selected on the aforementioned tabular component." —heavily paraphrased from CS-340 (n.d.-a, Prompt: §2, 5)

These tasks was already completed in a previous assignment (Ahmann, 2025b, §2.1–2.2). To briefly recap,[10] I have made changes to starter code presented for that assignment, specifically with adding and adjusting parameters in the `dash_table`,[11] and making changes to the given `update_map` function to display more information in the "tooltip" of the geolocation marker.[12] This formed the basis for the current project that is the subject of the next section.

3 Solution Presented: The Making of a Visual Data-Modelling Web Application

3.1 Task 3: The development of MongoDB queries by which to model, and filter, the `aac.animals` dataset

> "Next, you will make sure that the dashboard filter options can properly retrieve data from the database. Start by **developing**

[10]For a more comprehensive description of these solutions, I refer the reader to Ahmann (2025b).

[11]See Ahmann (2025b, Plate 1.3).

[12]See Ahmann (2025b, Plate 2.1).

	Rescue Type	**Preferred Breeds**	**Preferred Sex**	**Training Age**
1.	Water	Labrador Retriever Mix, Chesapeake Bay Retriever, Newfoundland	Intact Female	26 weeks to 156 weeks
2.	Mountain or Wild- -erness	German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky,	Intact Male	26 weeks to 156 weeks
3.	Disaster or Indiv- -idual Tracking	Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler	Intact Female	200 weeks to 300 weeks
4.	No filter	-	-	-

Table 1: Different kinds of rescues, and their respective "preferred attributes." After CS-340 (n.d.-b, p. 4).

database queries that match the required filter functionality. Refer to the Rescue Type and Preferred Dog Breeds Table, located in the Dashboard Specifications Document, to help you construct these queries."—CS-340 (n.d.-a, Prompt: §3)

To quickly identify the best candidate animals for a rescue mission, they need to have a set of preferred attributes, which include a preferred breed, preferred sex, and preferred age range. Table 3.1 depicts these attributes. The solution to this problem, as discussed in the prompt, involves writing MongoDB queries that return documents from the `aac.animals` collection conditional on criteria defined in table 3.1, in which the "rescue type" acts as a kind of query that returns a subset of the `aac.animals` collection that meets the criteria of its respective "preferred breeds," "preffered sex" and "training age."

Plate 1: Worked Filter for "Water Missions"

```
{
  "breed": {
    "$in": ['Labrador Retriever Mix',
      'Chesapeake Bay Retriever',
      'Newfoundland']
   },
   "sex_upon_outcome": 'Intact Female',
   "age_upon_outcome_in_weeks": {
     $gte: 26, $lte: 156
   }
}
```

To demonstrate a use case for data modelling for this project, I have worked out a MongoDB filter to select animals that meet the criteria for "water" rescue missions— as depicted by plate 1. This filter, along with other filters, are stored in a dictionary called `specific_queries`, and is referenced by a "key." These filters are then passed through the custom CRUD module's `read()` function, and the function returns BSON documents that meet the criteria embodied by the filter.

Appendix B shows additional filters for each of the different kinds of rescue missions that are to be used as an argument for the `read` function. In the case of the end-user's desire to select *all* animals regardless of their optimal predisposition for a rescue mission, an empty dictionary {} is passed as an argument to the `read()` function. Such filters will be used in the visual presentation of the datasets.

3.2 Task 4: The addition of components that allow for end-user selection of different data models

> “You must develop the controller pieces to **create interactive options that allow for the selection of data based on your filtering functions**. Develop these pieces in your IPYNB file, and be sure to import and use your CRUD Python module queries from Step 3. These interactive options will enable the **control of other dashboard widgets**, such as the data table and charts..”—CS-340 (n.d.-a, Prompt: §4)

As noted earlier,[13] some of this task was already completed in previous assignments. The work here is to introduce components that renders documents from the `aac.animals` collection based on a filtering criteria defined in the previous task. In my case, I decided to go with radio buttons. Plate 2 depicts the code snippet that I used to render radio buttons.

Plate 2: Worked code for “Radio Buttons” inserted into the `app.layout`

```
dcc.RadioItems(id = ’filter-type’,
  options = [
    { "label": "Water", "value": "water" },
    { "label": "Mountain or Wilderness",
 "value": "mt_or_wild" },
    { "label": "Disaster or Individual Tracking",
 "value":"dis_or_indv" },
    { "label": "No filter", "value": "no_filter" }
    ],
   value = "no_filter"
)
```

[13] In §3.1 of this writeup.

The important bit is the setting of options: defining individual radio buttons with “labels” to be rendered onto the screen, and a “value” that acts as an input.[14] When a particular radio button is selected, its `value` is then passed as a key to the `specific_queries` dictionary, which identifies the filter criteria to use for updating the table with the optimal animal candidates for a particular rescue mission. Plate 3 depicts the worked solution to the “middle man” callback used to update the datasets as new radio buttons are clicked on.[15]

Plate 3: Worked Solution to Dashboard Updater

```
1. @app.callback(Output('datatable-id','data'),
2.               [Input('filter-type', 'value')])
3. def update_dashboard(filter_type):
4.     data = pd.DataFrame.from_records(
5.         db.read(specific_queries[filter_type])
6.     )
7.     data.drop(columns=['_id'], inplace=True)
8.     return data.to_dict("records")
```

This code works by taking in a filter given the currently selected radio button,[16] then a new transaction of retrieving documents from the `aac.animals` collection on the condition of the filter for optimal candidates for rescue animals,[17] and finally returns the results as a dictionary.[18] This code will update the tables and charts in the current dashboard.

Two additional charts are given to assist the end-user in processing information from the `aac.animals` collection. The first is a geolocation map, which was already implemented in a previous assignment (Ahmann, 2025b, Plate 2.1), and the reader may consult that assignment’s writeup to learn more about the details. Here, I am interested in discussing the other chart that I implemented. While it was recommended that I implement some kind of pie chart to complement the geolocation chart, I instead decided to implement a *kernel density estimation* (KDE) chart of age distribution by “intact

[14]Plate 2, Lines 2–7.

[15]For the sake of academic honesty, I must credit Dengler (n.d.) for their indirect help: I got stuck with setting up this part of the project, and reviewed their solution to figure out where I was going wrong.

[16]Plate 3, Line 3.

[17]Plate 3, Lines 4–7.

[18]Plate 3, Line 8.

male/female" animals.[19] Plate 4 depicts the worked solution to the KDE chart component.

Plate 4: Worked Function to KDE Plotter

```
1. def update_graphs(viewData):
2.     dff = pd.DataFrame.from_dict(viewData)
3.     male_condition = dff[dff[
     "sex_upon_outcome"] == "Intact Male"]
4.     male_distribution = male_condition[
     "age_upon_outcome_in_weeks"]
5.     female_condition = dff[dff[
     "sex_upon_outcome"] == "Intact Female"]
6.     female_distribution = female_condition[
     "age_upon_outcome_in_weeks"]
7.     render = ff.create_distplot(
8.        [male_distribution, female_distribution],
9.        ["Age (Intact Males)", "Age (Intact Females)"],
10.    )
11.    render.update_layout(title_text="Age
     Distribution of Intact Males \nvs Intact
     Females by the Current Filter")
12.    return [
13.       dcc.Graph(figure = render)
14.    ]
```

The important code bits are the application of boolean filtering to select data entries,[20] declaring the data vectors for age by "intact maleness/femaleness,"[21] and the rendering of the chart with *Plotly*'s "figure factory."[22]

This represents the finished, worked solution to the basic data analysis web application: a front-end web application that displays information obtained from a back-end, non-relational database. These transactions between

[19]My decision to implement a KDE chart instead is because of my disdain for pie charts; as articulated in Diez et al. (2019, p. 66): pie charts are more difficult to interpret than other kinds of charts, such as bar charts, when analysing what per cent a part makes up of the whole.

[20]Plate 4, Lines 3 and 5.

[21]Plate 4, Lines 4 and 6.

[22]Plate 4, Lines 7–10, see *Distplots in Python* (n.d.). Retrieved on Oct. 20, 2025 from: https://plotly.com/python/distplot/

a client and server are, of course, mediated by an "in-house" Python CRUD module that demonstrates a basic level of reliability and functionality.

3.3 Task 6: Testing the recently invented, and further refined, web application and "CRUD Module"

> "Finally, after developing all of your code, you must test and deploy the dashboard to make sure that all of your components work. To complete this step, run your IPYNB file. You must either **take screenshots or create a screencast of your dashboard and widget functionality**. Each of your screenshots or your screencast should contain the **Grazioso Salvare logo** and your **unique identifier**."—CS-340 (n.d.-a, Prompt: §6)

To demonstrate that the web application is able to work as intended and as specified in the specifications document (CS-340, n.d.-b), I have ran it and recorded its behaviour in the form of screenshots and a screencast.[23] The screenshots demonstrating the dashboard are shown in Appendix A. In particular, the following screenshots given are depicted in:

- Figure 4: this is the dashboard with no filter.
- Figure 5: this is the dashboard with a "water rescue" mission filter.
- Figure 6: this is the dashboard with a "mountain or wilderness" mission filter.
- Figure 7: this is the dashboard with a "disaster or individual tracking" mission filter.

The dashboard returns different results from the database based on the criteria set. From the screenshots, specifically figure 4, **please note that there are error messages, of which I do not know its antecedents, nor do I have a solution for removing them at the moment**.

[23]The screencast is a supplementary material saved as an MP4 video footage.

4 End Matter

Unfortunately, due to time constraints, I could not explore, nor address, other topics that would act as a "stepping stone" for further development of this web application. Nonetheless, I have demonstrated a working proof-of-concept[24] for a visual data-modelling web application, which can interact with a backend non-relational database management system, implemented in Python's Dash framework. I have also produced documentation in the form of a `README` file to show how to use the web application, which acts as a condensed version of this writeup.[25]

4.1 Acknowledgements

Victor Udeh for the screenshot that they produced (Udeh, n.d.), which allowed me to better visualise what the resulting geolocation chart should look like. Joseph Dengler for their *Project 2* codebase (Dengler, n.d.), which I briefly referenced when I got stuck making the radio buttons work with the application's callback mechanism. And Professor Sanford for his guidance and thoughtful criticisms that I used as a basis for improvement.

References

Ahmann, A. (2025a). *CS-340: Assignment 5-1: Writeup to Project 1.*

Ahmann, A. (2025b). *CS-340: Assignment 6-1: Module 6 Assignment.*

Animal Services. (2016). Austin Animal Center Outcomes (Version 3.1). *City of Austin, Texas Open Data.* https://doi.org/10.26000/025.000001

CS-340 (n.d.-a). *Project Two Guidelines and Rubric.*

CS-340 (n.d.-b). *Dashboard Specifications Document.*

Dengler, J. (n.d.). *Project 2 FINAL for CS-340 Client/Server Development.* Retrieved on Oct. 20, 2025 from: https://github.com/JPDengler/Client-Server-Development-CS-340

Diez, D., Cetinkaya-Rundel, M., and Barr, C. D. (2019). *OpenIntro Statistics [Fourth Edition].* OpenIntro Series.

[24] Much more is needed in the realm of software testing and quality assurance: the use of linters, unit testers, fuzzers, and, depending on the "stakes" of the problem, provably correct systems, to ensure that this web application can be deployed to "the wild."

[25] In accordance with §7 of the prompt; see CS-340 (n.d.-a) prompt.

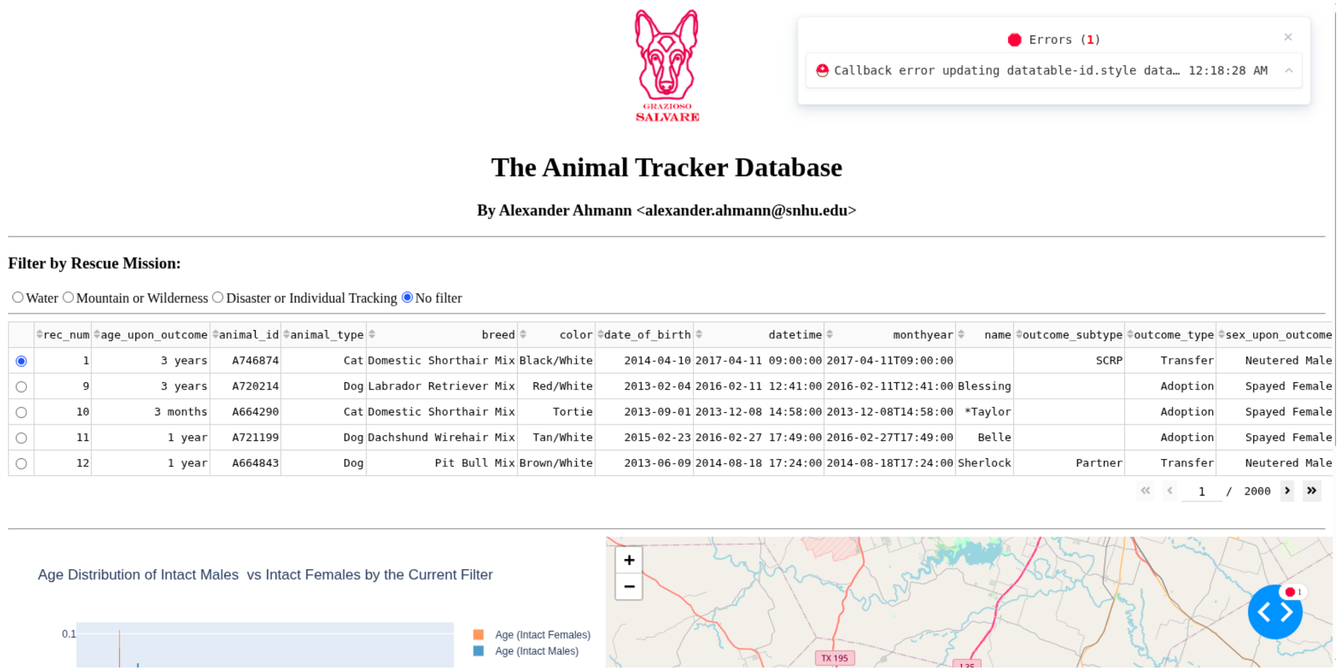
Lindemulder, G. & Kosinski, M. (n.d.). *What is role-based access control (RBAC)?*. IBM Think. Retrieved on Oct. 18, 2025 from: https://www.ibm.com/think/topics/rbac

Sandhu, R. S. (1997). Role-based Access Control. *In Advances in Computers (pp. 237–286).* Elsevier. https://doi.org/10.1016/s0065-2458(08)60206-5

Udeh, V. (n.d.). *Snhu-CS340 / module 6.* Retrieved on Oct. 12, 2025 from: https://github.com/vhicktour/Snhu-CS340/tree/main/module%206

van Rossum, G., Warsaw, B., & Coghlan, A. (Jul. 5, 2001). *PEP 8 — Style Guide for Python Code.* Python Enhancement Proposals. Retrieved on Oct. 18, 2025 from: https://peps.python.org/pep-0008/

A Figures Depicting Screenshots Demonstrating Project 2's Execution



Errors (1)

Callback error updating datatable-id.style data… 12:18:28 AM

GRAZIOSO SALVARE

The Animal Tracker Database

By Alexander Ahmann <alexander.ahmann@snhu.edu>

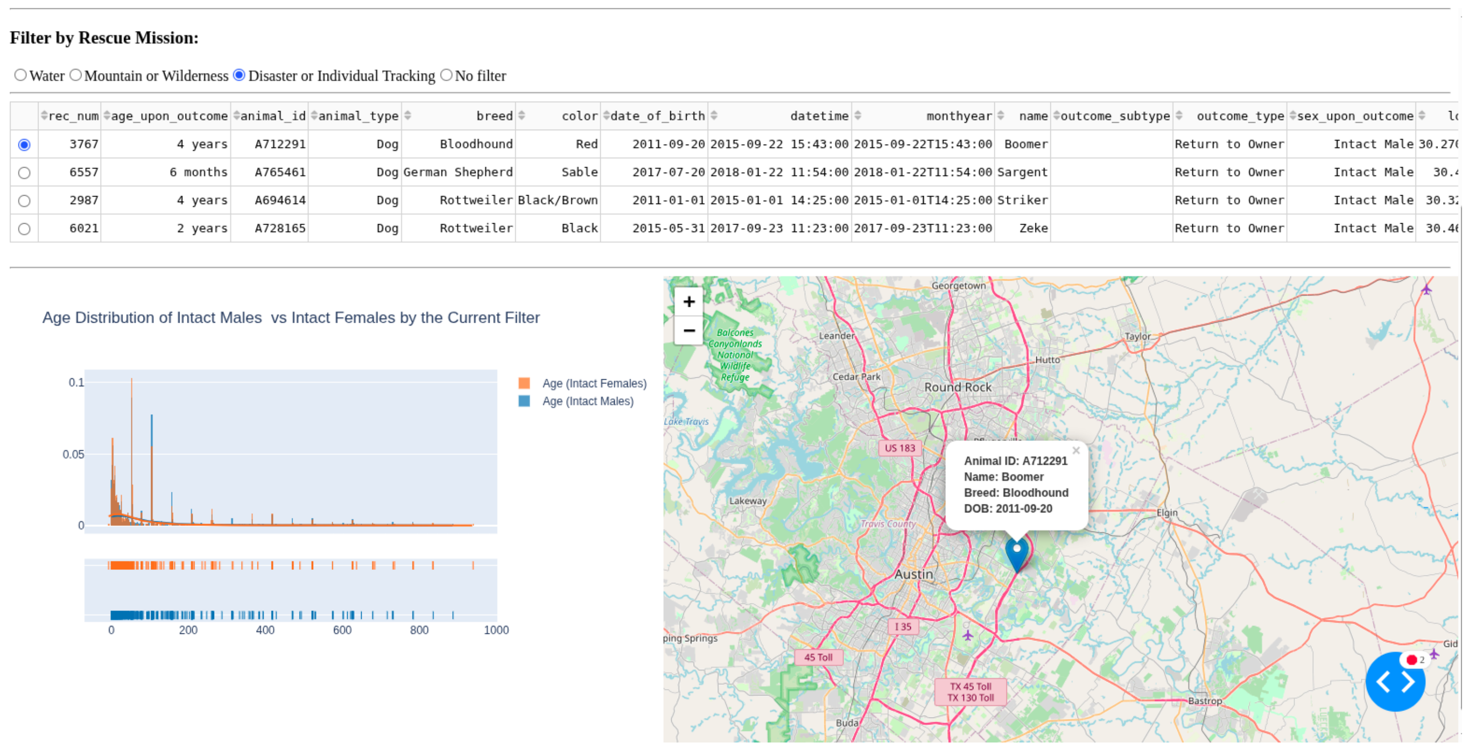
Filter by Rescue Mission:

Water Mountain or Wilderness Disaster or Individual Tracking [✓ No filter]

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome
●	1	3 years	A746874	Cat	Domestic Shorthair Mix	Black/White	2014-04-10	2017-04-11 09:00:00	2017-04-11T09:00:00		SCRP	Transfer	Neutered Male
○	9	3 years	A720214	Dog	Labrador Retriever Mix	Red/White	2013-02-04	2016-02-11 12:41:00	2016-02-11T12:41:00	Blessing		Adoption	Spayed Female
○	10	3 months	A664290	Cat	Domestic Shorthair Mix	Tortie	2013-09-01	2013-12-08 14:58:00	2013-12-08T14:58:00	*Taylor		Adoption	Spayed Female
○	11	1 year	A721199	Dog	Dachshund Wirehair Mix	Tan/White	2015-02-23	2016-02-27 17:49:00	2016-02-27T17:49:00	Belle		Adoption	Spayed Female
○	12	1 year	A664843	Dog	Pit Bull Mix	Brown/White	2013-06-09	2014-08-18 17:24:00	2014-08-18T17:24:00	Sherlock	Partner	Transfer	Neutered Male

« ‹ 1 / 2000 › »
Age Distribution of Intact Males vs Intact Females by the Current Filter
0.1
Age (Intact Females)
Age (Intact Males)
+
−
TX 195

Figure 4: Dashboard with No Filter.



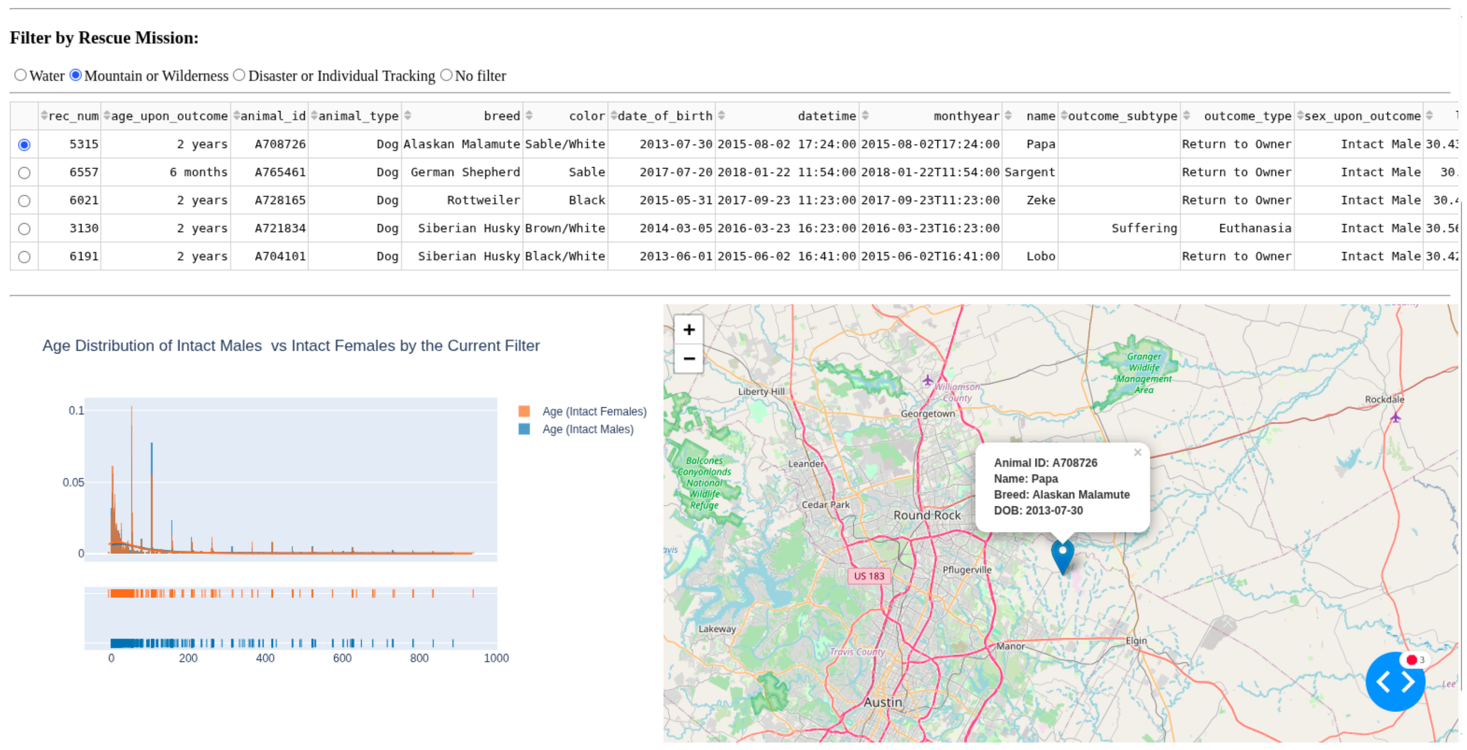
Filter by Rescue Mission:

Water Mountain or Wilderness [✓ Disaster or Individual Tracking] No filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome	lo
●	3767	4 years	A712291	Dog	Bloodhound	Red	2011-09-20	2015-09-22 15:43:00	2015-09-22T15:43:00	Boomer		Return to Owner	Intact Male	30.27
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.
○	2987	4 years	A694614	Dog	Rottweiler	Black/Brown	2011-01-01	2015-01-01 14:25:00	2015-01-01T14:25:00	Striker		Return to Owner	Intact Male	30.3
○	6021	2 years	A728165	Dog	Rottweiler	Black	2015-05-31	2017-09-23 11:23:00	2017-09-23T11:23:00	Zeke		Return to Owner	Intact Male	30.4

Age Distribution of Intact Males vs Intact Females by the Current Filter

Figure 5: Dashboard with “Water Rescue” Mission Filter.



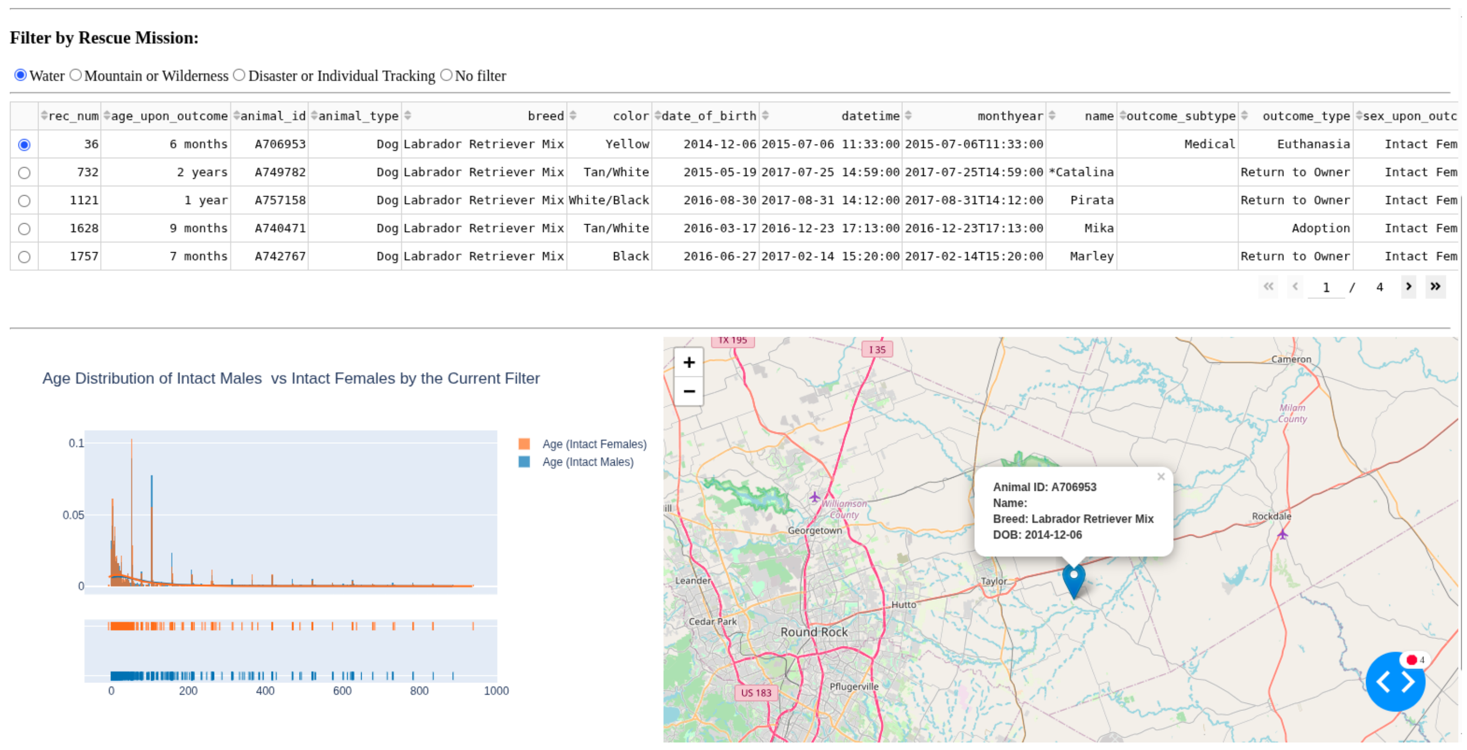
Filter by Rescue Mission:

Water [✓ Mountain or Wilderness] Disaster or Individual Tracking No filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome	
●	5315	2 years	A708726	Dog	Alaskan Malamute	Sable/White	2013-07-30	2015-08-02 17:24:00	2015-08-02T17:24:00	Papa		Return to Owner	Intact Male	30.4
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.
○	6021	2 years	A728165	Dog	Rottweiler	Black	2015-05-31	2017-09-23 11:23:00	2017-09-23T11:23:00	Zeke		Return to Owner	Intact Male	30.4
○	3130	2 years	A721834	Dog	Siberian Husky	Brown/White	2014-03-05	2016-03-23 16:23:00	2016-03-23T16:23:00		Suffering	Euthanasia	Intact Male	30.5
○	6191	2 years	A704101	Dog	Siberian Husky	Black/White	2013-06-01	2015-06-02 16:41:00	2015-06-02T16:41:00	Lobo		Return to Owner	Intact Male	30.4

Animal ID: A708726
Name: Papa
Breed: Alaskan Malamute
DOB: 2013-07-30

Figure 6: Dashboard with “Mountain or Wilderness” Mission Filter.



Filter by Rescue Mission:

[✓ Water] Mountain or Wilderness Disaster or Individual Tracking No filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outc
●	36	6 months	A706953	Dog	Labrador Retriever Mix	Yellow	2014-12-06	2015-07-06 11:33:00	2015-07-06T11:33:00		Medical	Euthanasia	Intact Fem
○	732	2 years	A749782	Dog	Labrador Retriever Mix	Tan/White	2015-05-19	2017-07-25 14:59:00	2017-07-25T14:59:00	*Catalina		Return to Owner	Intact Fem
○	1121	1 year	A757158	Dog	Labrador Retriever Mix	White/Black	2016-08-30	2017-08-31 14:12:00	2017-08-31T14:12:00	Pirata		Return to Owner	Intact Fem
○	1628	9 months	A740471	Dog	Labrador Retriever Mix	Tan/White	2016-03-17	2016-12-23 17:13:00	2016-12-23T17:13:00	Mika		Adoption	Intact Fem
○	1757	7 months	A742767	Dog	Labrador Retriever Mix	Black	2016-06-27	2017-02-14 15:20:00	2017-02-14T15:20:00	Marley		Return to Owner	Intact Fem

Animal ID: A706953
Name:
Breed: Labrador Retriever Mix
DOB: 2014-12-06

Figure 7: Dashboard with “Disaster or Individual Tracking” Mission Filter.

B Worked out Filters for the `aac.animals` Collection

B.1 Query for “Water” Rescue

```
{
  "breed": {
    "$in": ['Labrador Retriever Mix',
      'Chesapeake Bay Retriever',
      'Newfoundland']
   },
   "sex_upon_outcome": 'Intact Female',
   "age_upon_outcome_in_weeks": {
     $gte: 26, $lte: 156
   }
}
```

B.2 Query for “Mountain or Wilderness” Rescue

```
{
  "breed": {
    "$in": ['German Shepherd',
      'Alaskan Malamute',
      'Old English Sheepdog',
      'Siberian Husky',
      'Rottweiler']
   },
   "sex_upon_outcome": 'Intact Male',
   "age_upon_outcome_in_weeks": {
     $gte: 26, $lte: 156
   }
}
```

B.3 Query for “Disaster or Individual Tracking” Rescue

```
{
  "breed": {
    "$in": ['Doberman Pinscher',
      'German Shepherd',
      'Golden Retriever',
      'Bloodhound',
      'Rottweiler']
   },
   "sex_upon_outcome": 'Intact Male',
   "age_upon_outcome_in_weeks": {
     $gte: 20, $lte: 300
   }
}
```