# CS 340 README Amauri Hopewell

Contents

[CS 340 README Amauri Hopewell 1](#_Toc212414928)

[About the Project/Project Title 1](#_Toc212414929)

[Screen Recording and Demonstration of key features: 2](#_Toc212414930)

[Motivation 3](#_Toc212414931)

[Getting Started 3](#_Toc212414932)

[Installation 6](#_Toc212414933)

[Usage 7](#_Toc212414934)

[Customization 7](#_Toc212414935)

[Roadmap/Features (Optional) 8](#_Toc212414936)

[Contact 8](#_Toc212414937)

## About the Project/Project Title

Grazioso Salvara Animal Shelter Dashboard: This project develops a custom web dashboard for Grazioso Salvare, an international rescue animal training company. The dashboard interfaces with data from animal shelters in Austin, Texas, stored in a MongoDB database. It allows users to identify and categorize dogs suitable for search-and-rescue training based on specific profiles (e.g., age, breed, sex). The application follows an MVC pattern: MongoDB as the model, Dash for view and controller. The dashboard is built using Python and deployed via JupyterDash for easy testing and interactivity.

This project also leverages a reusable Python module to implement Create and Read steps of a Create, Read, Update, Delete (CRUD) operations system for a MongoDB database for the “Grazioso Salvare” animal shelter management system.

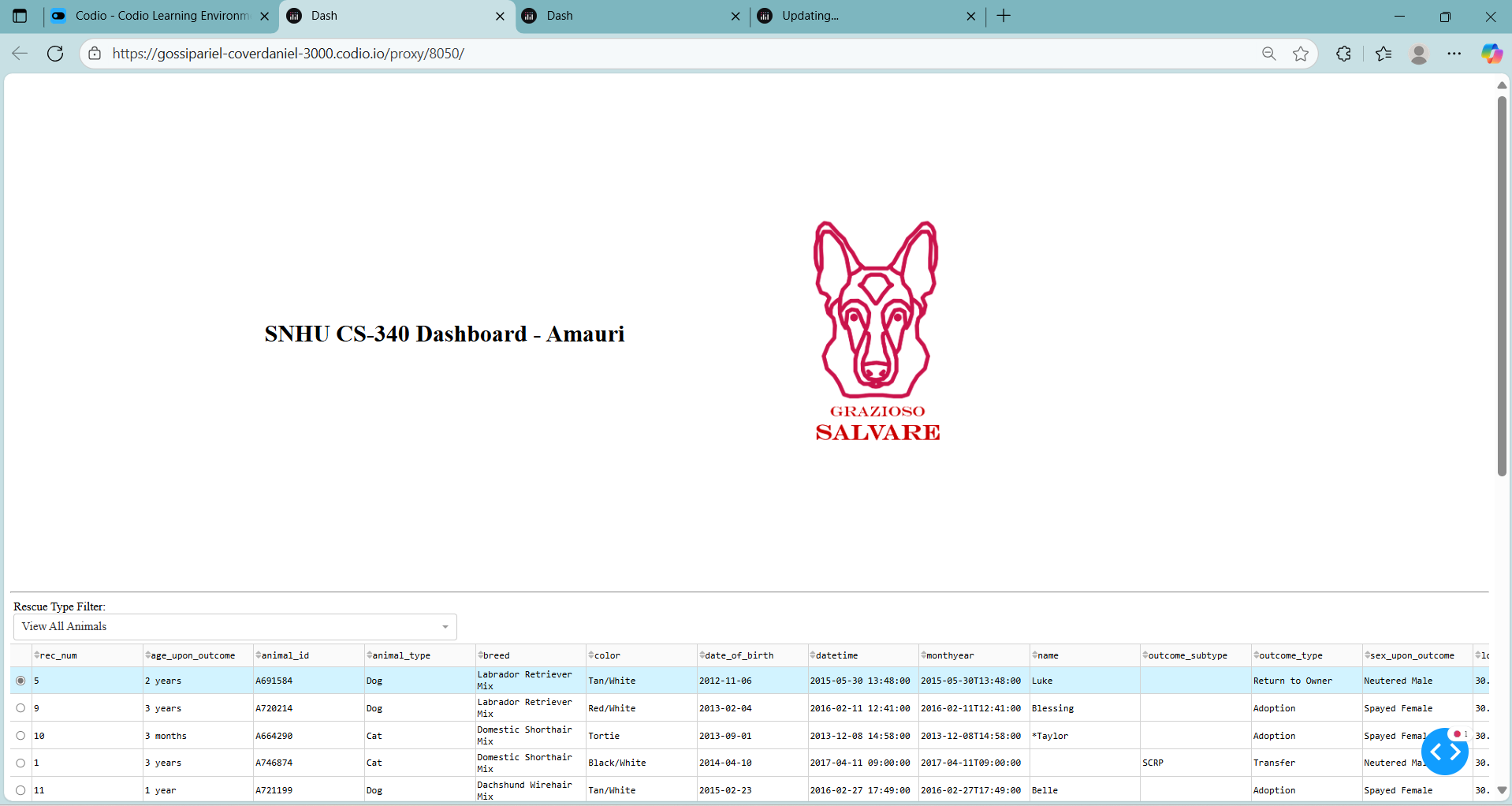
The AnimalShelter class provides object-oriented CRUD functionality to interact with the "aac" database and "animals" collection, enabling data inserting and querying animal records (such as breeds and age). It supports authenticating a user account ("aacuser") and is designed to be used in larger applications, such as Jupyter notebooks.

Key features:

* Interactive Filters: A dropdown menu to filter dogs by rescue type (Water Rescue, Mountain/Wilderness Rescue, Disaster/Individual Tracking) or reset to show all data. This dynamically updates the data table and charts based on predefined queries (e.g., specific breeds, age ranges in weeks, and sex).
* Data Table: An interactive table displaying filtered animal outcomes data, with pagination (10 rows/page), sorting, and single-row selection to update the geolocation chart.
* Geolocation Chart: A Leaflet map showing the selected animal's location, centered dynamically on the animal's coordinates (columns for latitude/longitude). Falls back to Austin, TX if data is invalid.
* Pie Chart: A breed distribution pie chart, showing top 5 breeds and "Other" for all remaining breeds, updating with filters.
* Branding: Includes the Grazioso Salvare logo (linked to www.snhu.edu) and a unique identifier ("Amauri").

## Screen Recording and Demonstration of key features:

The following shows the initial screen:



**This screen recording shows expanded functionality, but was too large to upload to github**

This functionality includes:

* Initial State (Reset Filter): Full unfiltered view of data table, pie chart with top breeds, map on default row.
* Water Rescue Filter: Table shows matching dogs (e.g., Labrador Retriever Mix, intact females aged 6-36 months); pie chart updates; map centers on selected animal.
* Mountain/Wilderness Rescue Filter: Similar, for breeds like German Shepherd, intact males aged 6-36 months.
* Disaster/Individual Tracking Filter: For breeds like Doberman Pinscher, intact males aged 5-69 months.
* Reset: Returns to full dataset.

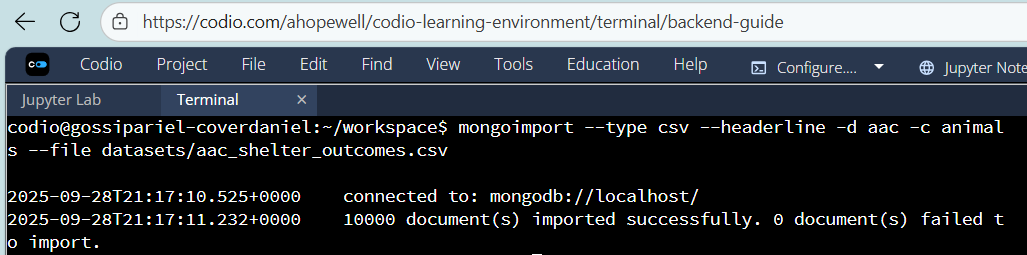
## Motivation

The Grazioso Salvare project streamlines animal shelter operations by using a NoSQL database instead of a relational database for flexible data storage. MongoDB is simpler, more intuitive, less error-prone, and more easily reusable.

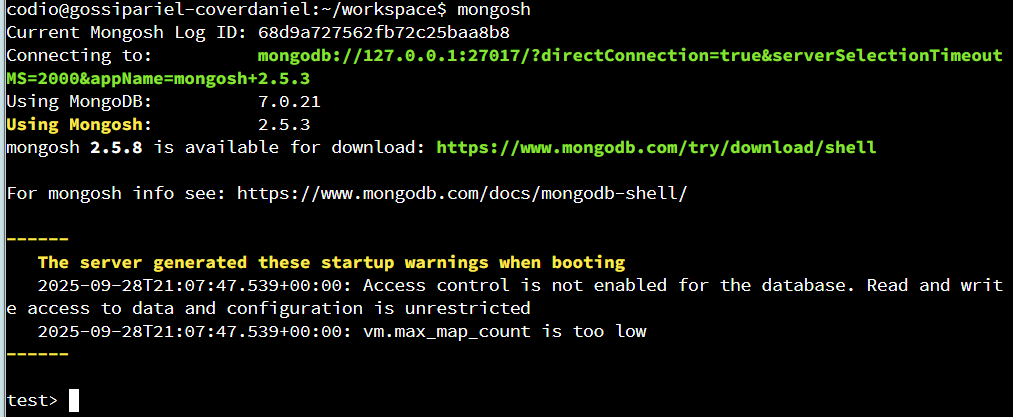
## Getting Started

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

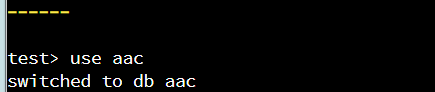
1. Setup the Database (from Module Three Milestone):
   * Import the "aac\_shelter\_outcomes.csv" dataset into MongoDB using mongoimport --type csv --headerline -d aac -c animals --file datasets/aac\_shelter\_outcomes.csv.



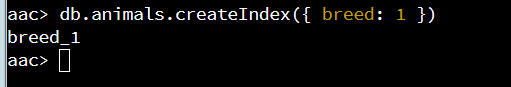
* + Then, connect to the database as shown:



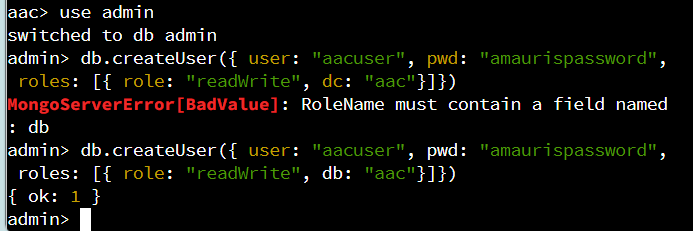
* + Next, switch to account AAC



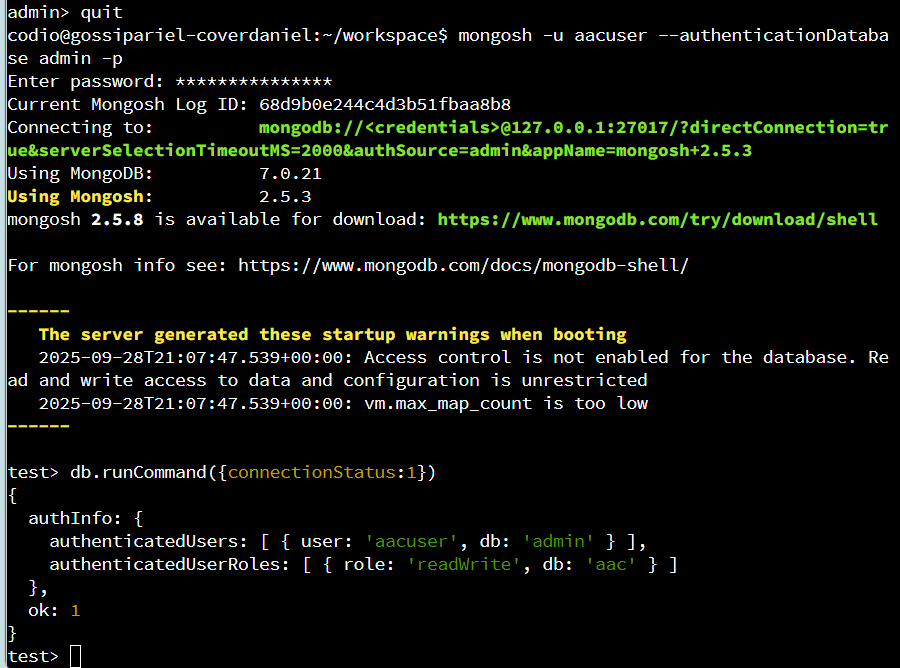
* + Next, create indexes: In mongosh, use aac; db.animals.createIndex({ breed: 1 }); db.animals.createIndex({ outcome\_type: 1, breed: 1 });.



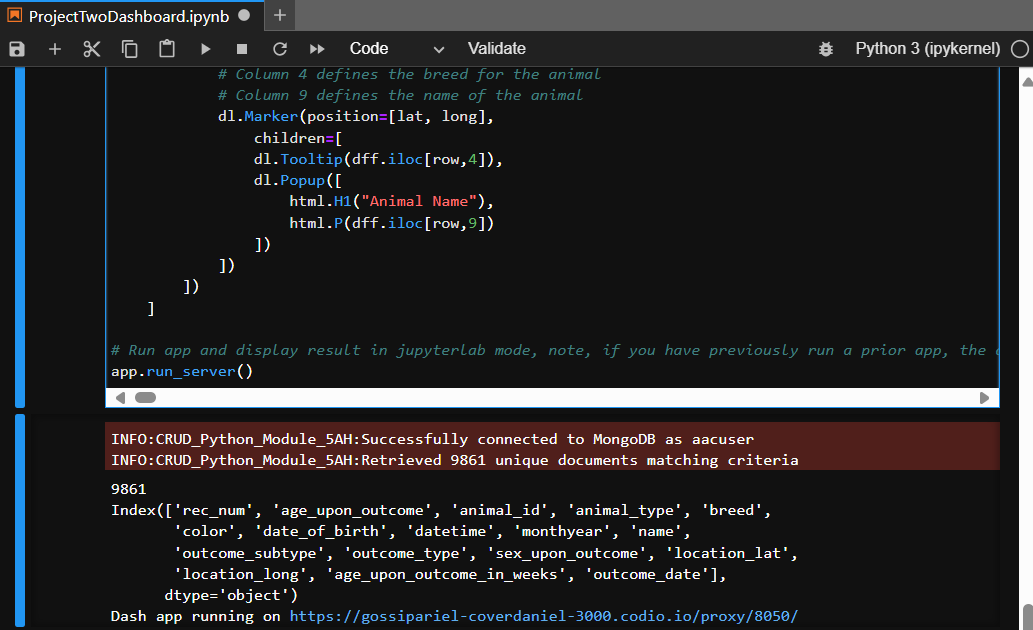
* + Set up "aacuser": In mongosh (as admin), use admin; db.createUser({ user: "aacuser", pwd: {your password}, roles: [{ role: "readWrite", db: "aac" }] }).



* + The login screen should show a result like the following:



1. Creating a Folder for your Python files
   * The folder will hold a .py and Jupyter notebook file
2. Running the Module:
   * Save .py file as CRUD\_Python\_Module\_5AH.py.
   * Save the Jupyter notebook and run it
   * It will show the address and port used to run the program, after “dash app running on,” as shown below:



* + Troubleshoot: Restart kernel for import changes
  + The following shows the successful test results from the test module:

## Installation

This Project has minimal dependencies, but does require

* Python 3.11: This is an easy-to-use, scripted programming language that allows rapid development and has extensive library and community support.
* MongoDB 6 (Available at MongoDB.com). This is used to create a no-SQL database for simplicity, allowing much easier management than relational databases such as SQL. MongoDB was selected for its NoSQL document-oriented structure, ideal for the flexible animal shelter data with varying fields like breeds and outcomes. It handles large datasets scalably without rigid schemas. It also comes with PyMongo, which integrates directly with Python, enabling seamless CRUD operations via dictionaries. Its most important capabilities include JSON-like documents for easy serialization, aggregation for deduplication, indexing for fast queries (as set up in milestones), and authentication support for secure access.
* Jupyter notebooks (Jupyter.org). This is used to test python files with short code snippets known as cells, making editing or testing a single step very easy and allowing very portable, easily-edited and modular code.
* Jupyter-Dash: This provides a framework for the web app with great integration with Jupyter notebooks, allowing easy testing while avoiding the hassle of Javascript.
* The python Logger library (install via pip if missing). This will allow logging messages to be displayed properly when handling errors and when integrating with Jupyter.
* Dash Leaflet: This provides support for the geolocation widget, bypassing the need for complicated Javascript coding or importing other libraries.

## Usage

These simple examples show the ease and power of this application:

**Example 1: View all animals (0:00-0:19 in screen recording)**

In its most basic functionality, this allows the user to view all animals and browse through a comprehensive paginated list, along with a pie chart summary of the breeds involved. The pie chart avoids excessive clutter by grouping less-frequently-occurring breeds into an “others” category. There is also a geolocation tracker, which automatically “snaps” to whatever animal is selected from the list.

**Example 2: Select animals for water rescue (0:19-0:40 in screen recording)**

This selects animals for the specific task of search-and-rescue in an aquatic environment. Thanks to the CRUD library’s newly-implemented deduplication feature along animal index, this shows a small number of individuals, selecting the most qualified for this specialized task, and resulting in selection of female labrador retriever mixes of a suitable training age.

**Example 3: Select animals for Mountain or Wilderness rescue (0:40-0:51 in screen recording).**

This selects animals for the specific task of search-and-rescue in rugged environments, selecting from rare but suitable breeds like the Siberian Husky or Alaskan Malamute. It implements the same functionality as example 1, but restricted to this smaller group of animals.

**Example 4: Disaster or Individual Tracking (0:51-1:01 in screen recording**

This selects animals suitable for tracking individuals that might be difficult to locate, and selects for breeds with a strong sense of smell such as bloodhounds.

**Example 5: Reset (1:01-1:25 in screen recording)**

This shows how a reset can be applied to view the original full list of animals with no loss in functionality.

## Customization

These were all implemented using a callback function for the dropdown menu functionality, which implemented the function update\_dashboard to create a new query called by the CRUD Module. The code snippet for this is below:

@app.callback(

[

Output('datatable-id', 'data'),

Output('pie-chart-id', 'children')

],

[Input('filter-type', 'value')]

)

def update\_dashboard(filter\_type):

if filter\_type == 'water':

query = {

"breed": {"$in": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]},

"sex\_upon\_outcome": "Intact Female",

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

}

elif filter\_type == 'mountain':

query = {

"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}

}

elif filter\_type == 'disaster':

query = {

"breed": {"$in": ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"]},

"sex\_upon\_outcome": "Intact Male",

"age\_upon\_outcome\_in\_weeks": {"$gte": 20, "$lte": 300}

}

else: # reset

query = {}

filtered\_data = shelter.read(query)

filtered\_df = pd.DataFrame.from\_records(filtered\_data)

if '\_id' in filtered\_df.columns:

filtered\_df.drop(columns=['\_id'], inplace=True)

By changing the dropdown menu options and adding the appropriate filters to the callback, any specialized task can easily be added to this dashboard.

## Roadmap/Features (Optional)

Future releases will add GUI support, provide troubleshooting tools, and allow connecting to multiple databases. In addition, there will be a custom filter creator, to allow adding custom tasks and filters to the dropdown menu without having to change the code, allowing Grazio Salvatore to maintain the system with less need for software developers.

## Contact

Amauri Hopewell (amauri.hopewell@snhu.edu)