# CS 340 - Project Two README

## Grazioso Salvare MongoDB Dashboard

## AAC Animal Outcomes: Interactive Dashboard with MongoDB CRUD Integration

## About the Project

This project is a Python-based application that interfaces with a MongoDB database to manage and visualize animal data from the Austin Animal Center (AAC). It builds upon the CRUD module developed in Project One, which provided core Create, Read, Update, and Delete (CRUD) operations through Python scripts, and extends that functionality into an interactive, client-friendly web dashboard.

The primary goal of this enhanced project is to make it easier for Grazioso Salvare to explore and analyze AAC animal outcome data through a visual and intuitive interface rather than raw database queries. The dashboard integrates directly with the MongoDB database to retrieve and display real-time information, allowing users to:

* Apply rescue-type filters (Water Rescue, Mountain/Wilderness Rescue, Disaster/Individual Tracking) to quickly find suitable animals for specific missions.
* Filter by animal type (Dog or Cat) to narrow results.
* View a dynamic data table with search, sort, and pagination features.
* Analyze breed distributions using a real-time pie chart.
* Identify the geographic location of animals through an interactive map with tooltips and popups.

By combining the robust data-handling capabilities of the CRUD module with the visual interactivity of Dash, this project streamlines the process of querying, interpreting, and presenting AAC data. It is designed for efficiency, usability, and scalability, ensuring that both technical and non-technical stakeholders can quickly access the insights they need for decision-making.

## Motivation

This project stems from the need to efficiently manage and interactively explore animal outcome data using a Python-driven interface. In Project One, the foundation was laid with a robust CRUD module, starting with Create and Read operations to streamline the insertion and retrieval of records. This ensured reliable database connectivity and query execution before expanding into Update and Delete functionality for full data lifecycle management.

In Project Two, this core data management capability has been extended into a fully interactive dashboard that not only handles data transactions but also transforms raw records into actionable insights. Instead of requiring users to write queries or sift through static reports, the dashboard provides real-time filtering, visualizations, and geolocation mapping, making it easier to identify animals that meet specific rescue criteria.

This evolution from a simple backend interface to a visually driven, end-user-friendly application reflects the project’s overarching goal: to make data access, interpretation, and decision-making faster, more intuitive, and more effective for Grazioso Salvare’s operational needs.

## Getting Started

To get started

1. MongoDB Configuration:
   1. A MongoDB server must be running with authentication enabled.
   2. An aacuser account must be created in the admin database with the readWrite privileges to the AAC database.
2. Authentication Setup
   1. When connecting, make sure to use correct aacuser credentials.
   2. Authentication database is set to admin.
3. Create and Read functionality
   1. Make a create\_animal() function that takes a python dictionary with details about the animal and inserts it into the collection.
   2. The read\_animal() function will retrieve documents based on provided filters.
   3. Create an update\_animal() function that modifies existing documents in the database.
   4. Create a delete\_animal() method that removes documents that match a given query from the collection.
4. Dashboard-Specific Additions

a. Add Rescue Type Filters (Water Rescue, Mountain/Wilderness Rescue, Disaster/Individual Tracking, Reset).  
b. Add Animal Type Filters (Dog, Cat, Reset).  
c. Create a DataTable with sorting, filtering, and pagination features.  
d. Add a Breed Distribution Pie Chart that updates dynamically based on filter selections.  
e. Add a Geolocation Map using dash\_leaflet to display animal locations.  
f. Include the Grazioso Salvare logo at the top of the dashboard linking to SNHU’s website.

1. Tip:
   1. Make sure the port and host values are correctly set in your python script.
   2. If using JupyterDash and you get a port in use error, change the port in app.run\_server(), restart your kernel or have ports be dynamically selected.

## Installation

Required tools:

Python – core programming language

MongoDB – NoSQL database used to store AAC animal data.

Jupyter Notebook- Used to test and demonstrate the module.

Pymongo – Python driver to interface with MongoDB

Dash / JupyterDash – Framework for building the interactive dashboard and running it inside Jupyter Notebook.

Plotly Express – Library for creating interactive visualizations such as the Breed Distribution Pie Chart.

dash\_leaflet – Library for rendering the interactive geolocation map in the dashboard.

Pandas – Data analysis library used to manipulate query results and feed them into visual components.

**MongoDB Configurations**

MongoDB Import and Setup Below we have and example on how to import your data.

To do this the mongoimport tool was used from the terminal. The command imported the aac\_shelter\_outcomes.csv file into the AAC database and the animals collection.

The import was configured to:

* Use authentication with the aacuser account
* Drop any existing data in the collection before reimporting
* Use the --type=csv flag to interpret the file format correctly
* Apply the --headerline option to treat the first row as field names

A screenshot of a computer

AI-generated content may be incorrect.

**MongoDB Authentication**

To secure access to the AAC database, a new user account named aacuser was created in the admin database using the mongosh shell. This user was granted read and write access to the AAC database. After creating the user, environment variables were set for simplified authentication:

export MONGO\_USER=aacuser

export MONGO\_PASS=your\_password\_here

Below are screenshots of what it looks like once you are properly authenticated.

A screen shot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.s

### Python Script to Connect To Database

**Import the following dependencies:**

from pymongo import MongoClient

from pymongo.errors import PyMongoError

**We then have to initialize the connection to MongoDB using valid credentials.**

class AnimalShelter:

def \_\_init\_\_(self, user='aacuser', password='password123', host='nv-desktop-services.apporto.com', port=32158):

**CRUD Methods**

**Create:**

def create(self, data):

"""

Insert a document into the MongoDB collection.

:param data: Dictionary containing the document to insert.

:return: True if successful, False otherwise.

"""

if data:

try:

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

return result.acknowledged

except PyMongoError as e:

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

return False

else:

raise ValueError("Data parameter is empty")

**Read**:

def read(self, query):

"""

Query documents from the MongoDB collection.

:param query: Dictionary with key-value pair(s) to use for lookup.

:return: List of documents (dicts) matching the query, or an empty list if none found.

"""

try:

cursor = self.collection.find(query)

return list(cursor)

except PyMongoError as e:

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

return []

**Update:**

def update(self, query, update\_data):

"""

Update documents matching the query.

:param query: Filter for documents to update.

:param update\_data: Dictionary of fields to update.

:return: Number of documents updated.

"""

if query and update\_data:

try:

result = self.collection.update\_many(query, {'$set': update\_data})

return result.modified\_count

except PyMongoError as e:

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

return 0

else:

raise ValueError("Query and update data must not be empty")

**Delete:**

def delete(self, query):

"""

Delete documents that match the query.

:param query: Dictionary of fields to match.

:return: Number of documents deleted.

"""

if query:

try:

result = self.collection.delete\_many(query)

return result.deleted\_count

except PyMongoError as e:

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

return 0

else:

raise ValueError("Query parameter must not be empty")

**Testing Script**

To test the python module I used a Jupyter notebook. Once you have setup your notebook you can run the following test:

**Connect to the Python Module:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Test Create Method with Expected Result**

**A computer screen shot of text

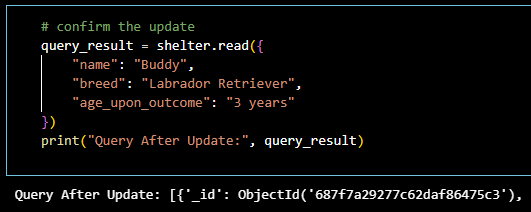
AI-generated content may be incorrect.**

**Test the Read Method**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**Test the Update Method**

****

**Test the Delete Method:**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

Below is a screenshot of what my Jupyter File looks like after testing.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Dashboard Configurations**

Dashboard Setup and Layout  
Below is an overview of how the Grazioso Salvare Dashboard was configured.  
The dashboard was developed using JupyterDash, which allows Dash applications to run directly inside a Jupyter Notebook. This setup ensures that all visual components, filters, and charts are dynamically updated based on user interaction.

The application integrates directly with the AnimalShelter CRUD module to perform queries against the MongoDB AAC dataset. Each query result is processed with Pandas and displayed in the dashboard’s components.

The dashboard consists of:

* A **clickable Grazioso Salvare logo** linking to the SNHU website.
* A **title** identifying the project and developer.
* A **Rescue Type filter** (radio buttons) to select:
  + Water Rescue
  + Mountain/Wilderness Rescue
  + Disaster/Individual Tracking
  + Reset (all records)
* An **interactive DataTable** with sorting, filtering, pagination, and row selection features.
* A **Breed Distribution Pie Chart** that updates dynamically according to active filters.
* A **Geolocation Map** that centers on Austin, TX, with markers, tooltips, and popups showing the selected animal’s breed and name.

**Dashboard Authentication**

The dashboard connects to MongoDB using the same aacuser account created in Project One. The CRUD module is instatiated with the required credentials:

A black screen with white text

AI-generated content may be incorrect.

Import necessary libraries:

A screen shot of a computer program

AI-generated content may be incorrect.

Load and Encode the logo and configure layout

A computer screen with many colorful text

AI-generated content may be incorrect.

Dashboard Interactivity

Filtering the Datatable based on Rescue Type or Animal Type:

A screen shot of a computer code

AI-generated content may be incorrect.

Dynamic Breed Distribution Pie Chart:

A computer screen shot of text

AI-generated content may be incorrect.

Interactive Map with markers and popups:

A computer code on a black background

AI-generated content may be incorrect.

Testing the Dashboard

Once the notebook is running, the dashboard loads inline. Testing steps include:

1. Verifying initial load of **all AAC data** in the DataTable.
2. Applying **each rescue type filter** and confirming that the DataTable, pie chart, and map update accordingly.
3. Selecting a row in the DataTable and confirming the map marker corresponds to the selected animal.
4. Resetting filters and confirming all data is displayed again.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a map

AI-generated content may be incorrect.

Dashboard with all the data loaded and pie distribution of all data including the logo.

A screenshot of a graph

AI-generated content may be incorrect.

Here we have a breed distribution on the pie chart of mountain/wilderness rescue.



A map with a blue point on it

AI-generated content may be incorrect.

The two screenshots above show the working geolocation after selecting an animal from our dashboard.

**Conclusion**

This project demonstrates the integration of MongoDB with Python using the PyMongo library to perform full CRUD operations on the Austin Animal Center (AAC) outcomes dataset. In Project One, we established the foundation by importing the AAC dataset, creating a secure authenticated database user, and developing a modular CRUD class to handle Create, Read, Update, and Delete functionality. This laid the groundwork for reliable, secure, and reusable database interactions.

Building on that foundation, Project Two extends the functionality into a fully interactive, client-facing dashboard using JupyterDash and related visualization libraries. This dashboard enables Grazioso Salvare to:

* Dynamically filter data by rescue type or animal type.
* Interactively search, sort, and page through results in a DataTable.
* View breed distribution trends through a dynamic pie chart.
* Explore animal locations with an interactive geolocation map.
* Quickly reset and reapply filters to explore the dataset from multiple perspectives.
* The project’s architecture follows the MVC design pattern, with:
* Model – MongoDB database containing AAC data.
* View – Dashboard components such as the DataTable, pie chart, and map.
* Controller – The CRUD Python module handling queries and data transformations.

By leveraging object-oriented programming principles, secure authentication, and modular code design, the system is both scalable and maintainable. Future enhancements could include additional filters, more detailed data visualizations, and integration with live data sources for real-time updates.

The completed solution equips Grazioso Salvare with a powerful decision-support tool, making it faster and easier to identify suitable animals for various rescue operations, ultimately improving mission planning and execution.

## Contact

Diego Caputi