# CS 340 Project Two README

**Developer: Jaden Knutson  
Client: Grazioso Salvare  
Email: Jaden.Knutson@SNHU.edu**

**About the Project/Motivation**

For this project, I’m the lead developer at Global Rain, working on a full-stack application for Grazioso Salvare, a company that trains rescue dogs in partnership with five animal shelters near Austin, Texas. They need a software tool to identify and categorize dogs for search-and-rescue missions based on factors like age and breed.

In Project One, my task was to build a MongoDB database with full CRUD functionality which consists of create, read, update, and delete. I completed the first phase by creating a Python module and Jupyter Notebook to show the CRUD operations using PyMongo. This database will later support a client-facing web dashboard and will be open source for broader use.

In Project Two, I created an interactive dashboard using Dash to enhance the database application. The dashboard provides an interface for shelter workers to use, filter, explore, and visualize possibilities for rescue animals, allowing better decisions and efficiency for rescue operations. I’ve also added abilities to highlight rows and columns, filter each category with text input, filter animals by rescue type, see a graph tied to the rescue data, filter categories with ascending and descending value, and see a map to see the location of the current selected animal

## Installation - The following tools and libraries were used:

* **Python:** This is the programming language for scripting the module.

Install from URL: <https://www.python.org/downloads/>

* **MongoDB:** This is a NoSQL database to store AAC animal data.

Install from URL: <https://www.mongodb.com/try/download/community>

* **PyMongo:** This is used to connect Python with MongoDB.

Install via terminal: pip install pymongo

* **Jupyter Notebook:** This Is used to test the module interactively.

Install via terminal: pip install notebook

## Getting Started - To get a local copy up and running, follow these simple steps:

1. Install and run MongoDB.
2. Set up the 'aacuser' account with password 'SNHU1234' on Apporto MongoDB instance.
3. Import aac\_shelter\_outcomes.csv dataset into MongoDB using the mongoimport tool.
4. Place animal\_shelter.py module in your project directory.
5. Create a Jupyter Notebook to import and test the class.
6. For Project two, add pip install dash, pip install jupyter-dash, pip install dash-leaflet, and pip install plotly.

## Usage - Code Example

**animal\_shelter.py -** This file defines the AnimalShelter class, which connects to a MongoDB database using PyMongo and uses pre-configured credentials for the Apporto environment.

It provides four methods (create, read, update, delete) for working with documents in the animals collection:

* **Method: create -** Adds a new animal document to the collection.

def create(self, data):

if data:

try:

self.collection.insert\_one(data)

return True

except Exception as e:

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

else:

print("Nothing to save, data is empty.")

return False

* **Method: read -** Obtains documents matching based on the query.

def read(self, query):

try:

return list(self.collection.find(query))

except Exception as e:

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

return []

* **Method: update – Updates all documents based on query and update values.**

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

try:

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

return result.modified\_count

except Exception as e:

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

return 0

* **Method: delete -** **Deletes matching documents based on the query.**

def delete(self, query):

try:

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

return result.deleted\_count

except Exception as e:

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

return 0

### Usage - Tests

**test\_crud.ipynb -** This file shows that the AnimalShelter class can correctly create, read, update, and delete data from the database.

from animal\_shelter import AnimalShelter

shelter = AnimalShelter("aacuser", "SNHU1234")

animal = {

"name": "Leonardo DiCATprio",

"animal\_type": "Cat",

"breed": "Maine Coon",

"outcome\_type": "Adoption"

}

# Create

shelter.create(animal)

# Read

for doc in shelter.read({"name": "Leonardo DiCATprio"}):

# Update

shelter.update({"name": "Leonardo DiCATprio"}, {"breed": "Siberian"})

# Delete

deleted\_count = shelter.delete({"name": "Leonardo DiCATprio"})

print(f"\nDeleted {deleted\_count} record(s).")

**Project 2:**

* **Radio Buttons and MongoDB Queries for Selecting Animal Rescue Type**

html.Label("Select Rescue Type"),

dcc.RadioItems(

id='filter-type',

options=[

{'label': 'Water Rescue', 'value': 'Water Rescue'},

{'label': 'Mountain or Wilderness Rescue', 'value': 'Mountain or Wilderness Rescue'},

{'label': 'Disaster or Individual Tracking', 'value': 'Disaster or Individual Tracking'},

{'label': 'Reset', 'value': 'Reset'}

],

value='Reset'

)

* **Creates an Interactive DataTable to view animals in the database.**

dash\_table.DataTable(

id='datatable-id',

columns=[

{"name": "Record #", "id": "rec\_num", "selectable": True},

{"name": "Age Upon Outcome", "id": "age\_upon\_outcome", "selectable": True},

{"name": "Animal ID", "id": "animal\_id", "selectable": True},

{"name": "Animal Type", "id": "animal\_type", "selectable": True},

{"name": "Breed", "id": "breed", "selectable": True},

{"name": "Color", "id": "color", "selectable": True},

{"name": "Date of Birth", "id": "date\_of\_birth", "selectable": True},

{"name": "Datetime", "id": "datetime", "selectable": True},

{"name": "Month/Year", "id": "monthyear", "selectable": True},

{"name": "Name", "id": "name", "selectable": True},

{"name": "Outcome Subtype", "id": "outcome\_subtype", "selectable": True},

{"name": "Outcome Type", "id": "outcome\_type", "selectable": True},

{"name": "Sex Upon Outcome", "id": "sex\_upon\_outcome", "selectable": True},

{"name": "Latitude", "id": df.columns[13], "selectable": True},

{"name": "Longitude", "id": df.columns[14], "selectable": True},

],

### Usage - Screenshots

Output test for create, read, update, and delete methods.

A screenshot of a computer

AI-generated content may be incorrect.

I created a new user aacuser with readWrite access to the aac database, set environment variables for the username and password, verified they were applied, and successfully logged in using mongosh. Running db.runCommand({ connectionStatus: 1 }) confirmed that aacuser was authenticated with the correct role.

**Step 1:** Import the CSV File Using mongoimport

A screen shot of a computer

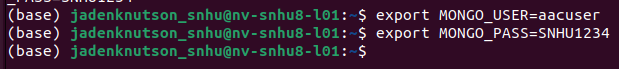
AI-generated content may be incorrect.

**Step 4:** Create aacuser Account

A computer screen with text

AI-generated content may be incorrect.

**Step 5:** Set Environment Variables



**Step 6:** Confirm Environment and Connect as User

A screen shot of a computer

AI-generated content may be incorrect.

**Step 7:** Run Connection Verification

A computer screen with white text

AI-generated content may be incorrect.

**Project 2:**

This shows the interactive dashboard the users can use to view and filter through the animal database.

**Example 1:** Using the Reset Filter.

A screenshot of a computer

AI-generated content may be incorrect.

**Example 2:** Using the Water Rescue Filter.

A screenshot of a computer

AI-generated content may be incorrect.

**Example 3:** Using the Mountain or Wilderness Rescue Filter.

A screenshot of a map

AI-generated content may be incorrect.

**Example 4:** Using the Disaster or Individual Tracking Filter.

A screenshot of a map

AI-generated content may be incorrect.

**Example 5:**

Shows the OTP pie chart and the map tags for the animals name and breed.

A map with different colored lines

AI-generated content may be incorrect.

**Challenges Encountered**

* One challenge I encountered was with adding the animal rescue types, where I initially used DataFrame functions for filtering instead of implementing the filtering through MongoDB queries. I had to redo my approach and create query dictionaries, then pass them to the read() method of the AnimalShelter class.
* Another challenge was designing the UI in a way that felt efficient. At first, the components were cluttered and didn't work well together visually. I spent time adjusting layout styles and spacing to make the dashboard more user-friendly, though there’s still room for improvement in this area.