**CS 340 README Grazioso Salvare Austin Animal Application**

**Overview**

The Grazioso Salvare Austin Animal Application is a full-stack software solution designed to identify suitable canine candidates for search-and-rescue training from five animal shelters in the Austin, Texas region. The application integrates a MongoDB database, populated with data from the Austin Animal Center (AAC) CSV file, and a client-facing web dashboard built using Jupyter Notebook, Dash, and Leaflet. The system includes a Python module for CRUD (Create, Read, Update, Delete) operations to manage animal data efficiently, enabling Grazioso Salvare to streamline the recruitment process for search-and-rescue canines.

**Table of Contents**

* Motivation
* Getting Started
* Installation
* Usage
* Features
* Tests
* Screenshots
* Contributing
* Contact

**Motivation**

This application enables Grazioso Salvare to efficiently identify and recruit ideal canine candidates for search-and-rescue training. As training must begin before a canine reaches two years of age, timely identification of suitable candidates from local shelters is critical. By leveraging data from the AAC database, this tool accelerates the adoption and training process, ultimately helping these canines save lives.

**Getting Started**

To begin using the Grazioso Salvare Austin Animal Application, ensure you have access to a Linux terminal with MongoDB installed and Jupyter Notebook for running the Python and Dash components. The application is hosted locally at localhost:6802/tree via Jupyter Notebook, where both the .py (core code) and .ipynb (testing and dashboard) files can be accessed.

**Prerequisites**

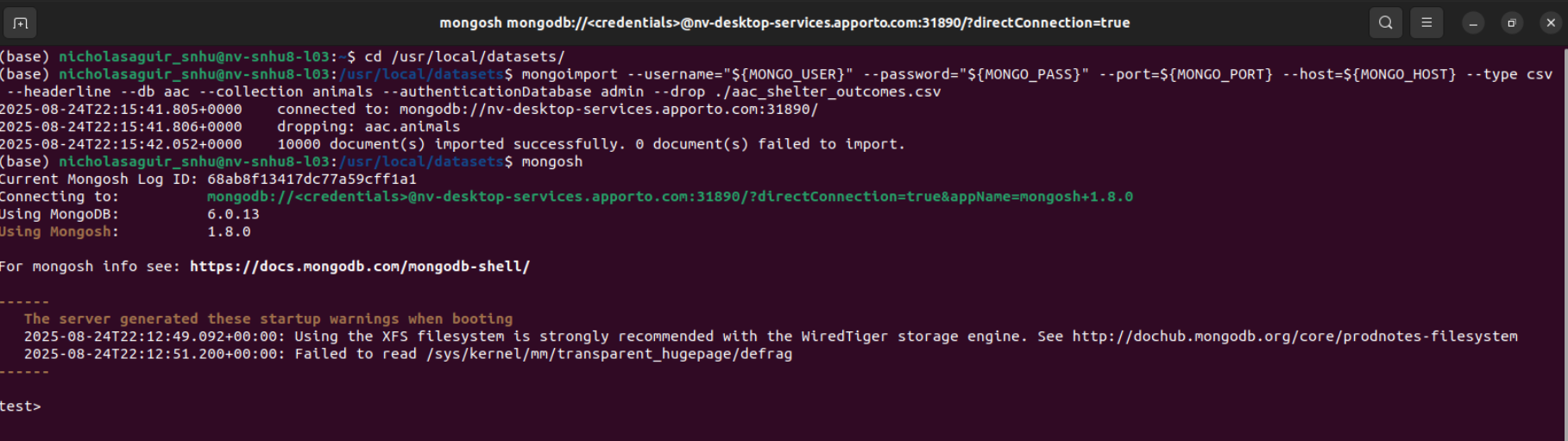
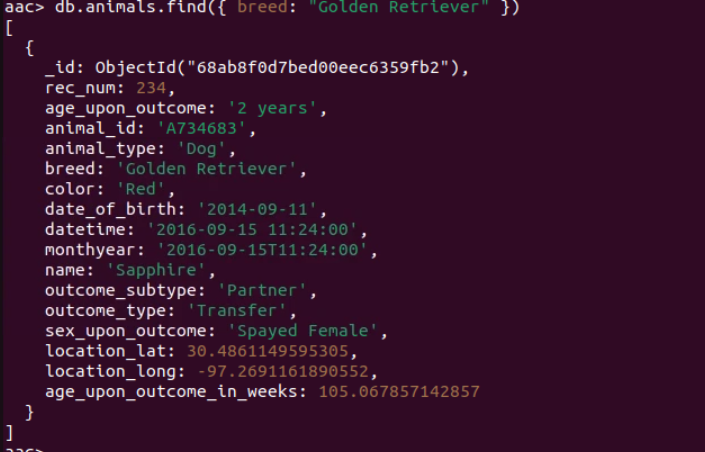
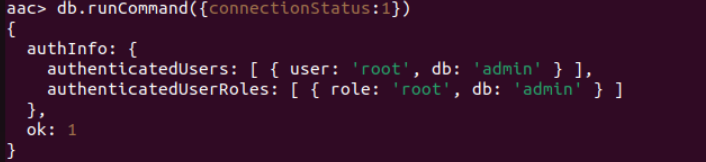
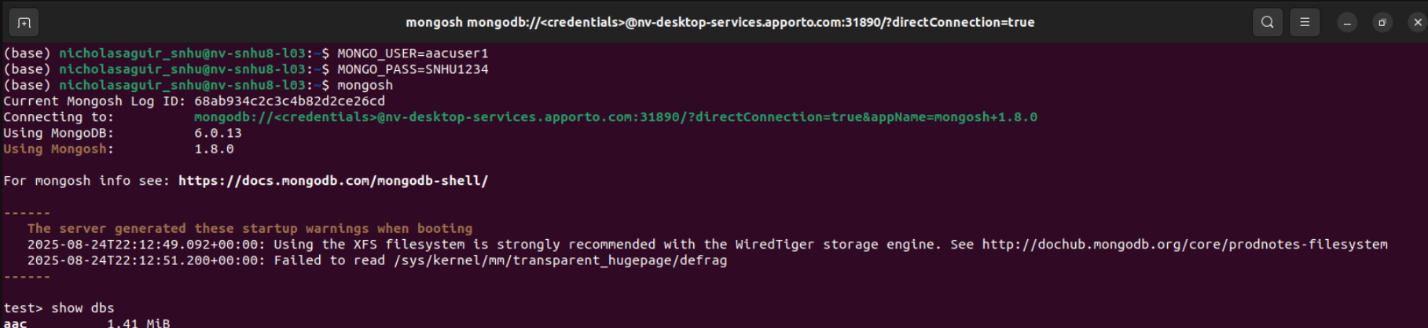
* MongoDB (mongosh)
* Python 3.6
* Jupyter Notebook
* Required Python libraries: pymongo, dash, dash-leaflet
* AAC CSV file (animal\_shelter.csv)
* User Authentication approved

**Product requirements requested by Grazioso Salvare**

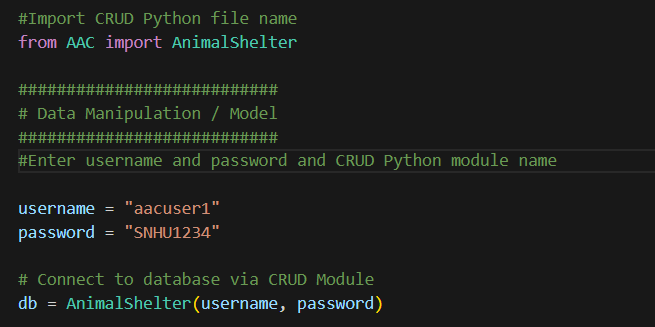
* + Grazioso Salvare logo and branding
  + Website link
  + Data table equipped with custom filters
    - Water rescue filter
    - Mountain or wilderness rescue filter
    - Disaster Rescue filter
  + Pie chart displaying available breeds
  + Geographic map displaying location of desired animals

**Installation**

Follow these steps to set up the project:

* **Import the AAC CSV file into MongoDB**:
  + Open a Linux terminal and import aac\_shelter\_outcomes.csv into MongoDB shell
  + Start MongoDB shell
  + 
  + Verify the import by querying the database in mongosh to find golden retriever
  + 
* **Set up user authentication for accessing aac database.**
  + Create admin account to authorize read/write privileges for user
  + 
  + Verify admin permissions are enabled in all databases including aac database.
  + Test authentication is successful by logging in MongoDb shell with new user and password.
  + New user can now access aac database

**Set up Jupyter Notebook**:

* + Install Jupyter Notebook if not already installed:
  + Use command pip install jupyter
  + Launch Jupyter Notebook:
* **Create and import Python files**:
  + Create a .py file (e.g., AAC.py containing the CRUD operations.
  + Import the .py file into an .ipynb file in Jupyter Notebook for testing and dashboard development
    1. 
* **Install required Python libraries**:
  + pip install pymongo dash dash-leaflet

**Usage**

The application allows authenticated users to interact with the AAC database via a Python CRUD module and a web-based dashboard. Below are details on how to use the system:

**CRUD Python Module**

The CRUD module (AnimalShelter class) is implemented in a Python file to manage the aac database. It uses the pymongo driver to connect to MongoDB, chosen for its seamless integration with Python and Jupyter Notebook, allowing for efficient data manipulation and testing.

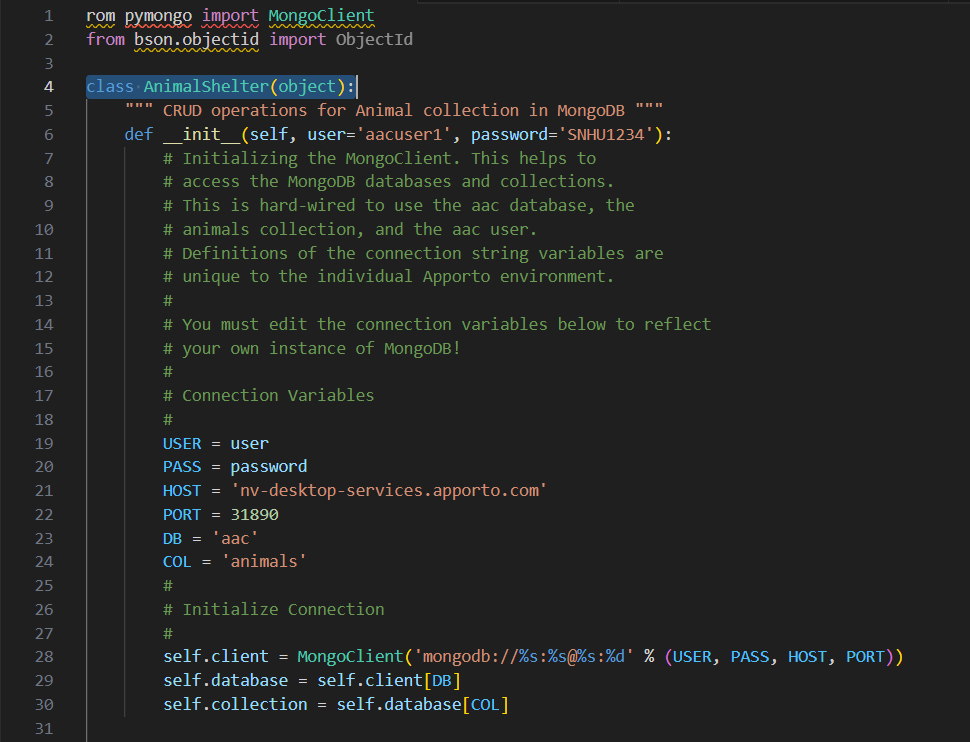
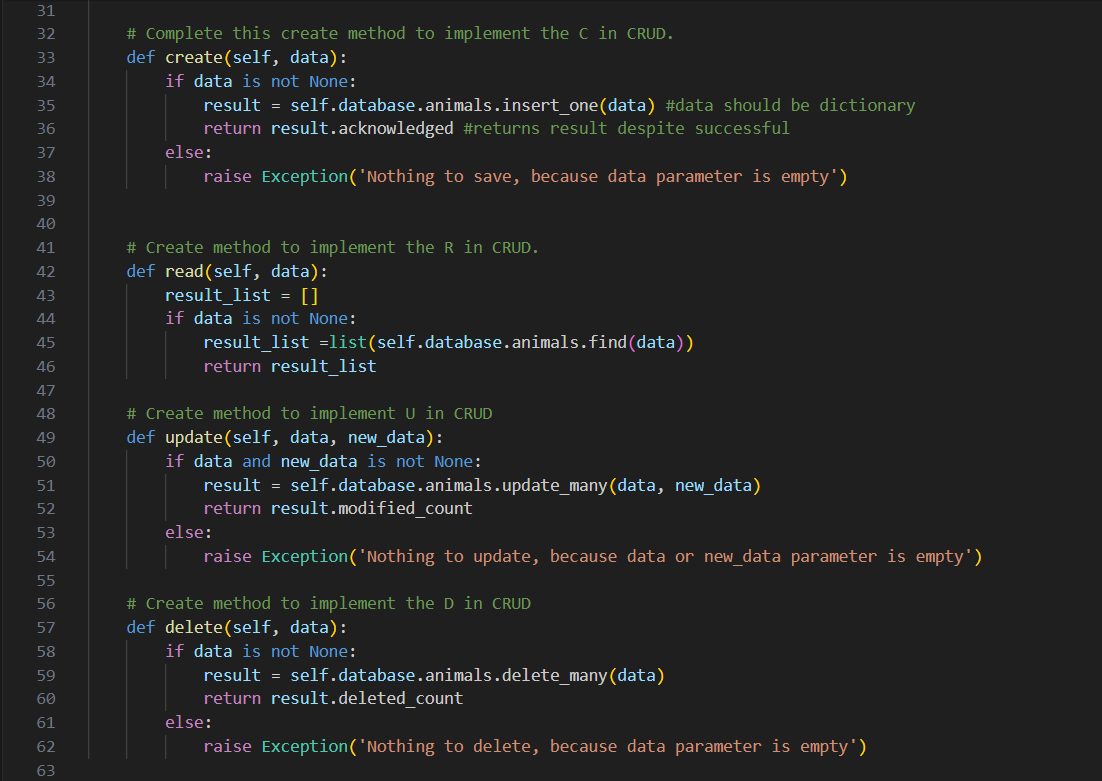
**Purpose**

The AnimalShelter class provides methods to create, read, update, and delete records in the aac.animals collection, enabling authorized users (e.g., aacuser1) to manage animal data effectively.

**Attributes and Functionality**

* **Create**: Adds new animal records to the database using attributes like animal\_id, name, and breed.
* **Read**: Retrieves animal data by specific attributes, returning all relevant information for the queried animal.
* **Update**: Modifies existing animal records to reflect changes, such as adoption status.
* **Delete**: Removes animal records when they are no longer available (e.g., adopted or deceased).
* **Authentication**: Access is restricted to authorized users (e.g., aacuser1) to ensure data security.

**CRUD Class and Constructors**



# Example usage

shelter = AnimalShelter('aacuser', 'password')

new\_animal = {'animal\_id': 'A123', 'name': 'Max', 'breed': 'German Shepherd'}

shelter.create(new\_animal)

result = shelter.read({'animal\_id': 'A123'})

print(result)

**Dashboard**

The dashboard, built with Dash and Leaflet, provides a visual interface for querying and displaying animal data. Access it via Jupyter Notebook at localhost:6802.

**Features**

* **Database Management**: CRUD operations for efficient management of animal records.
* **Interactive Dashboard**: Visualizes animal data with filters for breed, age, and other attributes.
  + User can select animal on left side via control map toggle
  + Pie Chart displays breed of dog and availability in shelters
  + Table with radio buttons acting as filters to fulfill Grazioso Salvare requirements
    - User can select **all**
    - User can select **water rescue**
    - User can select **mountain/wilderness rescue**
    - User can select **disaster rescue**
  + Geographic map showing location of selected animal using Leaflet
    - Map can be moved around using mouse equipped with zoom in and zoom out features for area of interest.
* **Secure Authentication**: Restricts database access to authorized users.
* **Scalable Design**: Built to handle large datasets from multiple shelters.

**Tests**

To verify the application's functionality, perform the following tests:

1. **Create**: Add a new animal record with animal\_id, name, and breed. Verify the record appears in the database:
2. shelter.create({'animal\_id': 'A999', 'name': 'Bella', 'breed': 'Labrador'})
3. result = shelter.read({'name': 'Bella'})

print(result) # Should display all attributes for Bella

1. **Read**: Query an animal by animal\_id and confirm all attributes are returned.
2. **Update**: Modify an animal's status (e.g., change status to 'Adopted'):

shelter.update({'animal\_id': 'A999'}, {'status': 'Adopted'})

1. **Delete**: Remove an animal record and verify it no longer exists:

shelter.delete({'animal\_id': 'A999'})

**Complications/Troubleshooting**

Upon completing the following application for Grazioso Salvare, some complications arouse in development. Getting the MongoDB to connect to the Jupyter notebook kernel was challenging with having to use the correct class and constructors. Wrong port was initially used and led to disconnect and inability to gain access to animal.csv file. Creating the dashboard with radio buttons was troublesome as each filter had to be selected only one at a time. Initial code had all filters automatically selected with any single selection. Overall, the project did not become overly complicated as all requirements required by Grazioso Salvare were manageable. Technology issues were related to user error and not setting up the dashboards and imports correctly. Learning Jupyter notebook was difficult at first but was conquered. Testing the CRUD functions in the ipynb file was doable. Data in the geographic map worked seamlessly with data table when filter was selected.

**Contact**

Nicholas Aguirre - email@example.com  
Project Repository: https://github.com/[username]/[repository]