# Grazioso Salvare Dashboard README

## About the Project

The Animal Shelter CRUD Python module is a backend component of the Grazioso Salvare project. It provides essential Create, Read, Update, and Delete (CRUD) operations for interacting with an animal shelter database stored in MongoDB.

In addition to database management, this project now includes an interactive dashboard that allows users to:

* Filter rescue dogs based on specialized criteria derived from the Austin Animal Center Outcomes dataset:
  + Water Rescue
  + Mountain/Wilderness Rescue
  + Disaster Rescue or Individual Tracking
  + Reset (restores default view)
* View a dynamic data table that updates based on selected filtering options.
* Interact with a geolocation map displaying animal locations.
* Explore a pie chart visualizing breed distribution within the filtered dataset (limited to top 10 breeds).
* Records are now sorted by rec\_num in ascending order, for consistent ordering.

## Motivation

The Grazioso Salvare project was created to support the mission of training rescue dogs to save lives in emergencies. By partnering with a nonprofit overseeing five animal shelters in Austin, Texas, this module:

* Streamlines identification of rescue dog candidates through structured data filtering.
* Analyzes factors such as age, breed, and outcome type to match dogs with ideal rescue roles.
* Provides an efficient way to manage shelter records using MongoDB and an interactive web interface.

**Steps Taken to Complete the Project**

1. **Database Design**: Set up MongoDB to store Austin Animal Center data, authenticate access securely, and ensure compatibility with Python.
2. **CRUD Implementation**: Developed Python methods for Create, Read, Update, and Delete operations.
3. **Interactive Dashboard**: Built a responsive dashboard using the Dash framework and Plotly for visualization, enabling filtering, sorting, and geolocation.
4. **Integration & Testing**: Ensured seamless integration of the database and dashboard. Verified functionality through a series of controlled CRUD tests.

## Getting Started

**Step 1: Install Prerequisites**

Before running the project, ensure you have access to the following tools:

* **MongoDB**: A NoSQL database that stores animal shelter records as JSON-like documents. Its dynamic schema and robust query capabilities allow flexible filtering and seamless integration with Python via Pymongo, supporting CRUD operations and efficient data handling for the dashboard.
* **Python 3.9.12**: Used to develop the backend module.
* **Pymongo**: Enables communication between Python and MongoDB.
* **Jupyter** Notebook: Used for testing and running CRUD operations and the dashboard.
* **Dash**: A Python-based framework for building interactive dashboards. It integrates the UI and controller logic, enabling dynamic updates to tables, charts, and maps based on user inputs, while leveraging Plotly for data visualization.
* **Plotly**: Supports data visualization (pie charts and geolocation mapping).

Refer to the Installation section for detailed setup instructions.

**Step 2: Set Up the Database**

* A MongoDB database named AAC was created to store the Austin Animal Center data. The data is stored in the animals collection and includes information such as animal type, breed, age, and outcome details.
* Authentication for the database was set up using the admin database in mongosh. A user account (aacuser) was created with restricted permissions to ensure secure access.

Refer to the Installation section for detailed steps on setting up the database

**Step 3: Add the CRUD Python Module**

* The CRUD functionality is implemented in the AnimalShelterCRUD class, which serves as the project’s controller for handling database interactions. The module provides:
  + Create: Insert new animal records into the database.
  + Read: Query the database for specific records.
  + Update: Modify existing records based on given criteria.
  + Delete: Remove records from the database.

Refer to the Usage section for detailed steps on interacting with the database using the module.

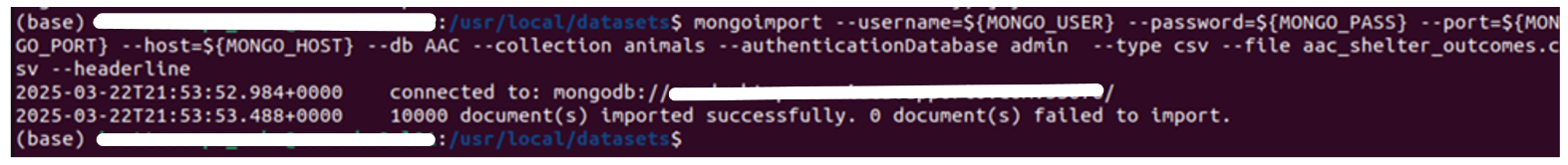
**Step 4: Run the Project**

1. Open Jupyter Notebook
   1. create a new notebook
   2. import the AnimalShelterCRUD module.
2. Test CRUD operations interactively by running queries against the database.

## Installation

**Disclaimer:** The following installation instructions are specific to a Linux operating system. If you’re using Windows or macOS, certain commands may differ.

1. **MongoDB** - Install MongoDB and start the service:
   1. sudo apt update
   2. sudo apt install -y mongodb
   3. sudo service mongod start
   4. Verify MongoDB is running:
      1. sudo service mongod status
2. **Python 3.9.12** - Install Python:
   1. sudo apt update
   2. sudo apt install -y python3.9
   3. python3 –version
3. **Pymongo -** Install the MongoDB Python driver:
   1. pip install pymongo
   2. pip list
4. **Jupyter Notebook -** Install Jupyter Notebook:
   1. pip install notebook
   2. jupyter notebook
5. **Dash and Plotly** 
   1. pip install dash
   2. pip install dash-leaflet
   3. pip install plotly
6. **MongoDB Dataset Import -** Import the provided dataset (AAC\_Austin\_Animal\_Center\_Outcomes.csv) into the MongoDB database:
   1. Replace ${MONGO\_USER}, ${MONGO\_PASS}, ${MONGO\_PORT}, and ${MONGO\_HOST}, with your specific MongoDB credentials and connection details.

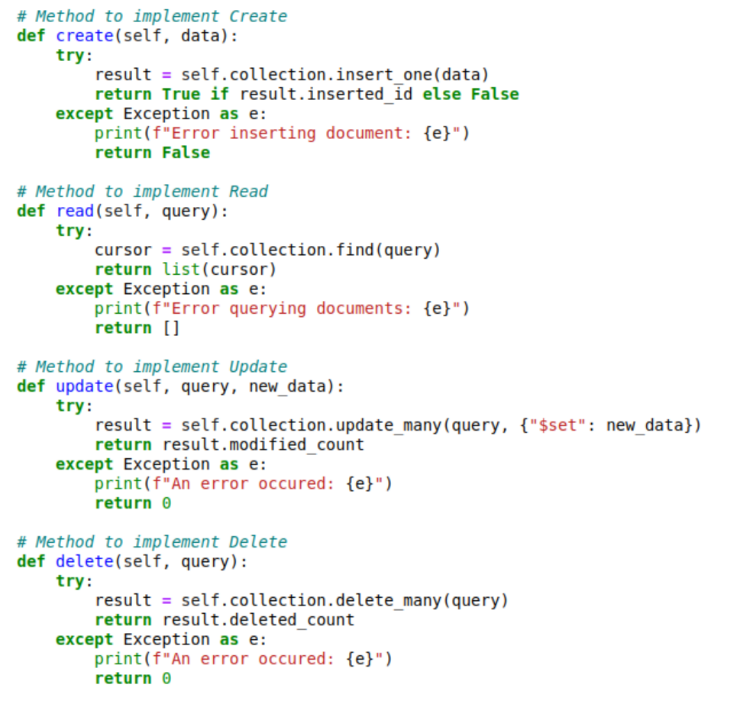


1. **Create Database User -** Open mongosh and create a user with restricted access for the database:
   1. Fill in user and pwd with credentials of your choice.

A computer screen shot of a program code

AI-generated content may be incorrect.

## Usage



**CRUD features**

1. Create:
   1. Purpose: Inserts a new document into the MongoDB collection.
   2. Implementation: Uses MongoDB’s insert\_one() method to add the provided data dictionary and returns True if successful.
   3. Error Handling: Includes a try-except block to catch insertion errors, which prints an error message and returns False for failure.
2. Read:
   1. Purpose: Retrieves documents matching the query.
   2. Implementation: Uses MongoDB’s find() method, returning a list of documents for ease of use in Python applications.
   3. Error Handling: Manages query errors with a try-except block, prints an error messages and returns an empty list in case of failure.
3. Update:
   1. Purpose: Modifies multiple documents that match the query.
   2. Implementation: Uses MongoDB’s update\_many() method with the $set operator to apply updates and returns the count of records modified.
   3. Error Handling: Wraps operations in a try-except block, prints an error message and returns 0.
4. Delete:
   1. Purpose: Removes multiple documents based on the query.
   2. Implementation: Uses MongoDB’s delete\_many() method, returning the count of deleted documents.
   3. Error Handling: Includes a try-except block, prints an error message, and returns 0 if an error occurs.

**Dashboard Features**

**A screenshot of a computer

AI-generated content may be incorrect.**

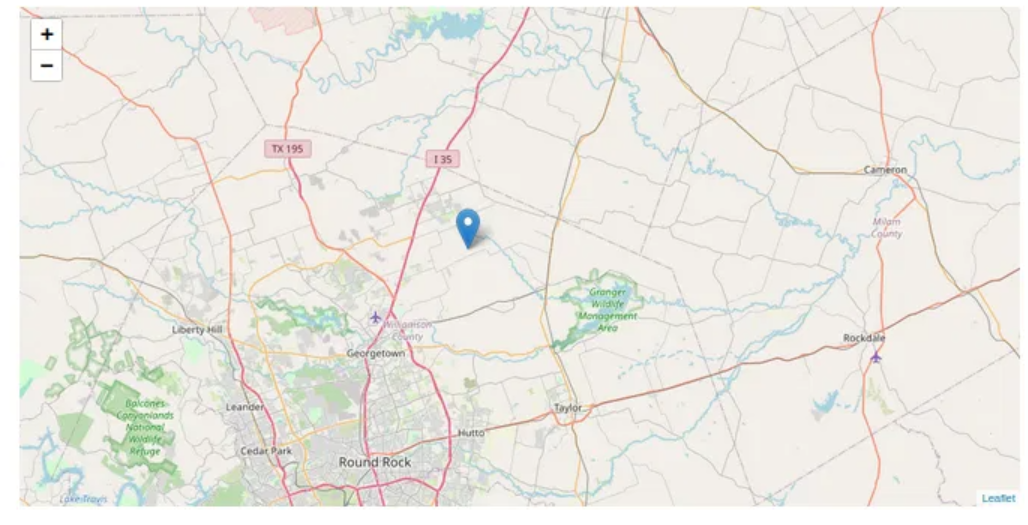
The interactive dashboard allows users to:

* Filter rescue dogs using radio buttons.



* A screenshot of a computer

  AI-generated content may be incorrect.View a sortable data table.
* Interact with a geolocation map (automatically selects first row if available).



* Analyze breed distribution via a pie chart (limited to top 10 breeds for better readability and sizing).

### A pie chart with different colored circles AI-generated content may be incorrect.

### Tests

1. Replace the username, password, host, and port fields with the credentials you set up to use
2. Then instantiate the CRUD class for the tests

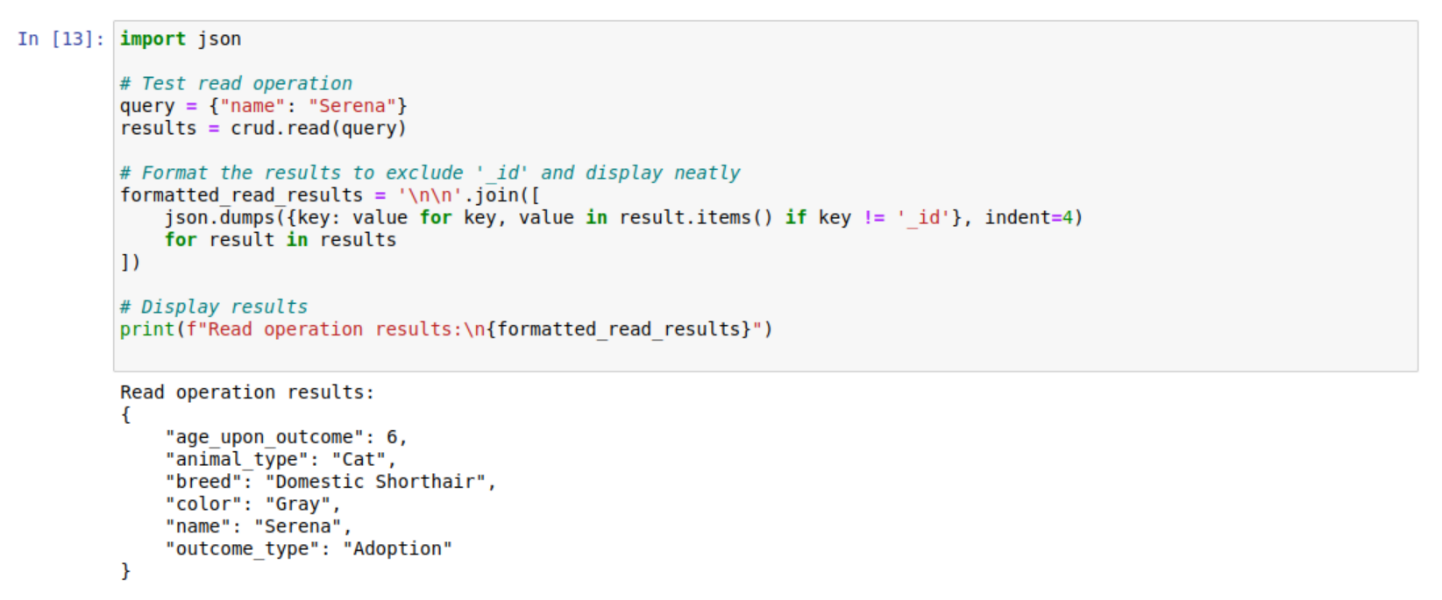


1. Then enter in the information about the animal to be added to the database. There are many fields that can be filled in but for the purpose of testing, I used minimal fields for this example. You should be looking for the Insert operation result to return True to indicate that the operation was successful.

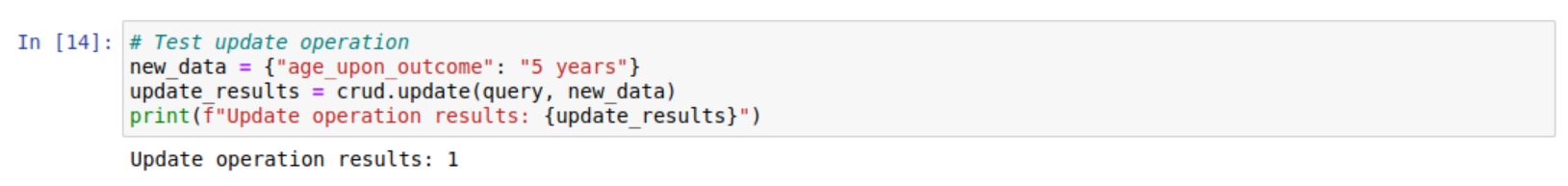
A screen shot of a computer

AI-generated content may be incorrect.

1. Next, we test the read operation. I used the name field for my query and then formatted the results for them to be easier to read (this required an import of json). The expected output here is a list of the results that match our query.



1. Next, we can test the update operation. In this example I wanted to update the age field from just a number to include a new age as well as the word year in that field. The expected output here is for the number of updated documents to be returned, which in this case was just one.

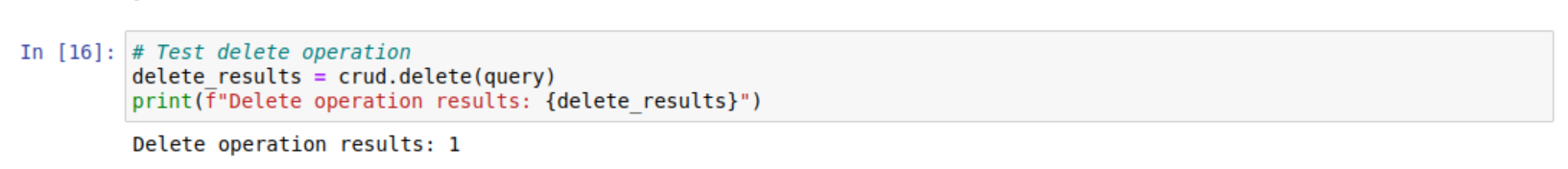


1. I wanted to verify that the document was really updated so I made another call to the read operation to see the updated age field.

A screen shot of a computer code

AI-generated content may be incorrect.

1. Next, I tested the delete operation. The expected output here is the number of documents in the database that are deleted based on our query. In this case it was just one document.



1. Finally, to verify that the document was deleted, I made another call to the read operation with our query and since we know there shouldn’t be a record with that query anymore, our expected output is an empty list.



**Challenges and Solutions**

1. **Handling MongoDB ObjectId Serialization**:
   * MongoDB's \_id field, represented as ObjectId, caused serialization issues when displaying data in Dash. This was resolved by converting the \_id field to a string and removing it from the dataset when unnecessary, ensuring compatibility with Dash's JSON requirements.
2. **Ensuring Consistent Data Sorting**:
   * The initial dataset and filtered results displayed inconsistent orders due to MongoDB's default retrieval logic. Explicit sorting by rec\_num in ascending order was applied at both the initial data retrieval step and within the filtering callback function.
3. **Reset Button Behavior**:
   * The Reset button cleared filters but retained previously selected rows, causing mismatches between displayed data and row selection. The callback logic was updated to reset both the dataset and the selected row index to the first row ([0]).
4. **Dashboard Interactivity with Filters**:
   * Updating the dashboard dynamically with Dash callbacks occasionally caused errors when datasets were empty or formatted incorrectly. Validating data before updates ensured proper structure and avoided issues.
5. **Initial Data Visualization Sizing**:
   * The pie chart and map displayed improperly sized visualizations upon loading. Explicit default sizes were set (e.g., 600x600 pixels for the pie chart) to maintain consistency.

## Contact: Ivette Cerpa