Grazioso Salvare AAC Dashboard

Table of Contents

* [About the Project](#_About_the_Project_2)
* [Development](#_Python_1)
* [MongoCrud](#_MongoCrud_(mongo_crud.py)_1)
* [Dash Dashboard](#_Dash_Dashboard)
* [Functionality](#_Functionality_1)
* [Tools](#_Tools_1)
* [Contact](#_Contact)

## 

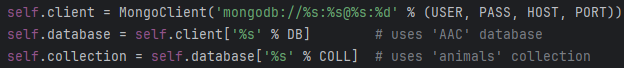
## About the Project

*This project utilizes a Model-View-Controller (MVC) software design pattern, with a Dash dashboard (View), a MongoDB database (Model), and callback functions (Controller). Databases are used in almost every type of software application, and MongoDB is among the most popular. MongoDB is a* [*NoSQL*](https://en.wikipedia.org/wiki/NoSQL) *(Not only SQL) document-oriented database system that specializes in horizontal scaling and fast transactions. Working with MongoDB can prove to be quite tedious, and it is common for users to make mistakes when working with a command-line interface (CLI)**.*

*The Dashboard for Grazioso Salvare is designed to interact with the Austin Animal Center (AAC) ‘animals’ collection, which holds data for cats and dogs that the AAC currently houses. The dashboard integrates with the custom mongo\_crud.py module’s MongoCrud class to abstract away the tedious nature of connecting to a* [*MongoDB*](https://www.mongodb.com/) *database and executing basic CREATE, READ, UPDATE, and DELETE (*[*CRUD*](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete#:~:text=In%20computer%20programming%2C%20create%2C%20read,basic%20operations%20of%20persistent%20storage.)*) operations. The user can create new documents to be inserted into the ‘animals’ collection of the’ AAC’ database, view (read) documents in the database by using custom query filters, update documents, and delete documents based on matching queries. More on functionality later.*

## Development

*The connection is initialized, and the user is authenticated, upon successful execution of the program. The mongo\_crud module creates a MongoClient class instance as seen below:*

**

*The module’s ‘create’ function performs a MongoDB ‘insert’ command:*

A screen shot of a computer code

Description automatically generated

*The module’s ‘read’ function performs a MongoDB ‘find’ command:*

A screen shot of a computer code

Description automatically generated

*The module’s ‘update’ function performs a MongoDB ‘update\_many’ command:*

*A screen shot of a computer program

Description automatically generated*

*The module’s ‘delete’ function performs a MongoDB ‘delete\_many’ command:*

*A screen shot of a computer code

Description automatically generated*

## MongoCrud (mongo\_crud.py)

*The mongo\_crud module’s MongoCrud class can be thought of as the middle “glue” layer between the MongoDB database (base level) and web app client that uses the “DASH” dashboard on the frontend.*

*The mongo\_crud.py module should be moved to the site-packages location that contains other 3rd party modules:*

* ***Windows****:* 
* ***macOS****/****Linux****:*

* OR *

* ***virtual environment*** *(like* [*Anaconda*](https://www.anaconda.com/)*):*

**

*Be sure to replace <username> with your username and ‘XX’ (or X.X) with your Python version.*

*PyMongo is a library with tools for working with MongoDB and is the official MongoDB driver for Python. PyMongo allows for server administration and even indexing to speed up database operations.*

*The mongo\_crud module’s class (MongoCrud), as well as MongoClient from the PyMongo library, needs to be imported into the API being developed, as seen here:*

**

**

*You would also need to install* [*PyMongo*](https://pymongo.readthedocs.io/en/stable/) *on your system, using* [*pip*](https://pypi.org/project/pip/) *or another installation method:*

**

*Now you can programmatically call the class methods from the module, based on the specific triggering mechanism(s) used in your given API or web application. Just create a MongoCrud instance:*

**

*To create a new document and insert the document into the database collection:*

**

*To access a document or group of documents by key-value pair(s):*

**

*To access the database, the user must be authenticated using a user account found in the ‘admin’ database. The ‘admin’ database uses* [*SCRAM*](https://en.wikipedia.org/wiki/Salted_Challenge_Response_Authentication_Mechanism)*-*[*SHA-256*](https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm) *encryption for password authentication. Currently, the user credentials are hard-coded in the dash web app; however, these strings containing credentials will be removed before deployment of the application.*

*The create and read functions start by ensuring the input value is not empty and is of the dict data type; otherwise, an exception is thrown:*

**

**

*In the create function, if the data is indeed of type* [*dict*](https://www.w3schools.com/python/python_dictionaries.asp) *(dictionary), then the new document is inserted into the collection by using the MongoDB ‘insert\_one’ function.*

*The read function uses a dictionary as a query filter by using the MongoDB ‘find’ function to retrieve the document as a Cursor object. I spent a bit of time struggling to find an efficient way to iterate through the Cursor object so that the queried results could be returned. To solve this problem, I ended up using a list comprehension that only took a single line of code:*



*If you plan on running these tests yourself (in the test\_mongo\_crud.py file), just make sure the mongo\_crud module is installed, as well as the Python* [*unittest*](https://docs.python.org/3/library/unittest.html) *library:*

**

*To run the included tests (assuming test\_mongo\_crud.py is located in site-packages), open your OS’s CLI, and follow the steps listed below:*

1. *Navigate to the site-packages folder where your third-party libraries should be installed.*
2. *Enter the following command in your CLI:*



*If you would like to run the tests programmatically from your API, you can simply import the test script and unittest, and execute it with code:*

**

**

*All tests should pass without error:*

*A pink and white striped background

Description automatically generated*

## Dash Dashboard

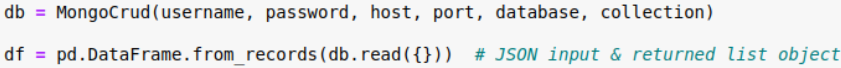
*The GUI (graphical user interface) dashboard was created using* [*Dash*](https://dash.plotly.com/) *from the* [*Plotly*](https://plotly.com/) *library, along with the* [*Python*](https://www.python.org/) *programming language and* JetBrain’s [PyCharm](https://www.jetbrains.com/pycharm/) integrated development environment (IDE) and[*Jupyter Notebook*](https://jupyter.org/)*.*

*To reproduce the dashboard, the appropriate libraries must be imported:*

*A screen shot of a computer

Description automatically generated*

*An instance of the MongoCrud class is instantiated by passing the information needed to connect to the ‘animals’ database and authenticate the user. Note that the mongo\_crud module has been updated to accept all connection details as passed parameters. Instantiation and document retrieval is seen below:*

**

*A JuptyerDash instance is also created and the Grazioso Salvare logo is attempted to be loaded.* *In addition to checking for empty inputs and implementing standard error-catching strategies, thorough testing was done on each method in the module to ensure a robust and secure product:*

*A computer screen shot of a code

Description automatically generated*

*To create the View, a layout is created:*

**

*The Controller consists of a series of dynamically reactive callbacks, which process changes in user input and output the appropriate changes:*

**

**

A screenshot of a computer

Description automatically generated



A map with a location pin

Description automatically generated

A map with a location on it

Description automatically generated

## Functionality

*The mongo\_crud module abstracts away the tedious nature of working with a MongoDB database. The user may sort and filter the table contents or limit the number of rows (documents) displayed. A pie chart initially appears, showing the animals with the highest occurrence at the Austin Animal Center:*

A screenshot of a computer

Description automatically generated

*In addition, a user may select one of the displayed filter options to show preferred dog breeds (and sexes and ages) preferable for training as rescue dogs of each specified type:*

**Water Rescues**

A screenshot of a computer

Description automatically generated

*As different rescue dog types are selected, the displayed table contents and pie chart data dynamically change to reflect the corresponding option:*

**Mountain or Wilderness Rescues**

A screenshot of a computer

Description automatically generated

*The ‘Reset’ button may be clicked to set options to their default state. Selecting an animal (row) will show a geolocation map, with the breed and name of the selected animal:*

**Disaster Rescues or Individual Tracking**

A screenshot of a map

Description automatically generated

## Tools

*[](https://www.mongodb.com/)*

*[A blue and white logo

Description automatically generated](https://www.python.org/)*

*[A white text on a black background

Description automatically generated](https://www.jetbrains.com/pycharm/?var=1)*

[](https://www.anaconda.com/)

[](https://pypi.org/project/pip/)

[](https://pymongo.readthedocs.io/en/stable/)

[A white letter on a black background

Description automatically generated](https://plotly.com/)

[](https://jupyter.org/)

## Contact

*Matthew Pool*

*Computer Science, Southern New Hampshire University*

[*matthew.pool@snhu.edu*](mailto:matthew.pool@snhu.edu)

*April 22, 2024*