# CS 340 README Template

## About the Project/Project Title

Grazioso Salvare Animal Shelter Application – Create, Read, Update, Delete (CRUD) Python Module

Grazioso Salvare, an innovative international rescue-animal training company, identifies dogs that are good candidates for search-and-rescue training. When trained, these dogs can find and help rescue humans or other animals, often in life-threatening conditions. To help identify dogs for training, Grazioso Salvare has reached an agreement with a non-profit agency that operates five animal shelters in the region around Austin, Texas. This non-profit agency will provide Grazioso Salvare with data from their shelters.

The purpose of this README file is to familiarize users/organizations that wish to use this software with details of the Create, Read, Update, and Delete (CRUD) Python module that was developed for the Grazioso Salvare Animal Shelter Application.

## Motivation

One of the primary reasons for the development of this application is because it was discovered that Grazioso Salvare looks for certain profiles in dogs to train. For instance, search-and-rescue training is generally more effective for dogs no more than two years old. Also, certain breeds of dogs are proficient at different types of rescue, such as water rescue, mountain or wilderness rescue, locating humans after a disaster, or finding a specific human by tracking their scent.

Grazioso Salvare is seeking a software application that can work with existing data from the animal shelters to identify and categorize available dogs. Grazioso Salvare has contracted Global Rain for a full stack development of this application, including a database and a client-facing web application dashboard. Grazioso Salvare has also requested that the code for this project be open source and accessible on GitHub so similar organizations may use and adapt it.

The Python module that was developed for this project is used to connect a client-side user interface (such as a dashboard) to a database which enables CRUD functionality for this data connection.

## Getting Started

To use this CRUD Python module, follow these simple steps:

* In a terminal window, import the Austin Animal Center (AAC) Outcomes data set.
* From your terminal, change the directory to the location of the AAC dataset.
* For example, type the following command: cd /usr/local/datasets (this should be the location of where you have stored the dataset locally).
* Then, type the following set of commands, please note that the commands are case sensitive.
  + mongoimport –username=”${MONGO\_USER}” \
  + --password=”${MONGO\_PASS}” --port=${MONGO\_PORT} \
  + --host=${MONGO\_HOST} --db AAC --collection animals \
  + --authenticationDatabse admin --type=csv --headerline \
  + --file=aac\_shelter\_outcomes.csv
* Press ENTER after each backslash and to execute the entirety of the command.
* Run the Mongo Shell as an administrative user. Type “mongosh” to execute the Mongo Shell.
* Access the “admin” database and create a new user called “aacuser” for the database “AAC”. The “aacuser” should have a readWrite role assigned to only the “AAC” database. This user is used to handle the direct interaction between the database and the Python module that is used to programmatically access data in the database.
  + In the Mongo Shell, execute the following commands to create the aacuser for the AAC database that will be used with our Python module.
    - Type “use admin” to switch to the admin database.
    - Type “db.createUser ( { user: “aacuser”, pwd: “SNHU1234”, roles: [ { role: “readWrite”, db: “AAC” } ] } )” and press enter to execute the command. (Note: all fields are case-sensitive)
* To verify that the user was successfully created, exit the Mongo Shell by typing quit.
* Type the following commands in the terminal to switch to the “aacuser” user.
  + MONGO\_USER=aacuser
  + MONGO\_PASS=SNHU1234
* Type “mongosh” to execute the mongo shell.
* Type the following command to see information about the authenticated user and their roles:
  + db.runCommand({connectionStatus:1})
* Type “show dbs” to verify that the current user only has access to the AAC database. Note: the only database that should be returned from this command is the AAC database.
* Exit the mongo shell and close the terminal.
* From a text editor of your choice open the animal\_shelter.py file for a review of the Python module and its CRUD functionality.
* Ensure the connection variables on lines 21-26 of the Python module are correct for your MongoDB connection.
* To test the functionality of the Python module, create a new jupyter notebook and save it as “animal\_shelter.ipynb” and your animal\_shelter.py file must be saved in the same location as your jupyter notebook file. Be sure to select Python 3 (ipykernel) when creating the new notebook.
* You can now begin executing commands to import the Python module, instantiate the AnimalShelter class, and utilize the CRUD operations for the AAC database from the Python module.

## Installation

The following is a list of tools needed to use the software and how to install them:

* The latest stable version of MongoDB – Installation tutorials can be found at MongoDB's website by following this link: [Install MongoDB — MongoDB Manual](https://www.mongodb.com/docs/manual/installation/)
* Retrieve the Austin Animal Center (AAC) Outcomes data set from: <https://doi.org/10.26000/025.000001>
* Python version 3.9 or better. This can be downloaded from <https://python.org/downloads>
* Jupyter Notebook – instructions for installing Jupyter Notebook can be found at <https://jupyter.org/install>
* Python editor or text editor of your choice. This will be used for viewing the Python module and editing it if you choose. The text editor used during the development of this project was FeatherPad 1.0.1 for Linux and can be retrieved at: <https://github.com/tsujan/FeatherPad/releases>

## Usage

### Code Example

When the AnimalShelter class is called for instantiation, the object that is created is initialized with connection variables then the connection is initialized with the specified client, database, and MongoDB collection. For example:

def \_\_init\_\_(self):

# connection variables

USER = ‘aacuser’

PASS = ‘SNHU1234’

HOST = 'nv-desktop-services.apporto.com'

PORT = 32435

DB = 'AAC'

COL = 'animals'

# Initialize Connection

self.client = MongoClient('mongodb://%s:%s@%s:%d' % (USER, PASS, HOST, PORT))

self.database = self.client['%s' % (DB)]

self.collection = self.database['%s' % (COL)]

Then, also within our AnimalShelter class, we have our CRUD methods that can be called to Create, Read, Update, or Delete Documents within the specified database and collection.

# create method

def create(self, data):

if data is not None:

try:

insert\_result = self.database.animals.insert\_one(data) # data should be dictionary

return True

except Exception:

return False

else:

raise Exception("Nothing to save, because data parameter is empty")

# read method

def read(self, readData):

if readData is not None:

data = self.database.animals.find(readData)

else:

raise Exception("Nothing to read, because data parameter is empty")

return data

# update method

def update(self, searchData, updateData):

if searchData and updateData is not None:

data = self.database.animals.update\_many(searchData, { "$set": updateData } )

else:

raise Exception("Nothing to update, because one of the data parameters is empty")

return data.modified\_count

# delete method

def delete(self, deleteData):

if deleteData is not None:

data = self.database.animals.delete\_many(deleteData)

else:

raise Exception("Nothing to delete, because the data parameter is empty")

return data.deleted\_count

### Tests

The Python module can be tested in the Jupyter Notebook you previously created during the Getting Started portion of this README file. We can test the module by doing the following:

In the Jupyter Notebook starting from the first line:

1. From animal\_shelter import AnimalShelter

- This will import the Python module.

1. animals = AnimalShelter()

- This creates and instantiates an animal object.

1. data = { 'age\_upon\_outcome': "5 years",

'animal\_id': "A55555",

'animal\_type': "Dog",

'breed': "American Bulldog",

'color': "White",

'date\_of\_birth': "2019-05-05",

'datetime': "2000-05-05 18:00:00",

'monthyear': "2000-05-05 18:00:00",

'name': "Buddy",

'outcome\_subtype': "",

'outcome\_type': "Adoption",

'sex\_upon\_outcome': "unknown",

'location\_lat': 32.1234567891234,

'location\_long': -95.1234567891234,

'age\_upon\_outcome\_in\_weeks': 30.1234567891234

}

- This creates a Python dictionary called data that will be used to insert and read a document in

the specified MongoDB database and collection.

1. print(animals.create(data))

- This inserts the document into the specified MongoDB database and collection.

- A result of true will be returned and displayed to you if the document is successfully

inserted.

- A result of False will be returned if the document does not get inserted into the database.

- An exception is thrown if a None data type is passed into the create method.

1. query = animals.read({'name': "Buddy",

'date\_of\_birth': "2019-05-05",

'outcome\_type': "Adoption" })

for animal in query:

print(animal)

- This calls the read method in the Python Module. If the data is found it is assigned to query

and a for loop is used to iterate through the query and display the data to the user.

1. print(list(animals.read({'name': "BuddyBuddyBuddy"})))

- This shows that when a query is made that does not match any documents in the specified

MongoDB and collection, an empty list is returned and displayed to the user.

1. print(animals.update({‘name’:”Buddy”,

‘date\_of\_birth’: “2019-05-05”,

‘outcome\_type’: “Adoption”},

{‘name’: “BuddyBuddy”}))

-This updates any document in the specified MongoDB database and collection based on the

passed-in parameters. The first parameter is the data that is being searched for. The second

parameter is the data that we want to update if any documents match the search data. The

number of documents updated is returned to the user. If no documents are updated, zero is

returned.

1. print(animals.delete({‘name’: “BuddyBuddy”}))

-This deletes any documents in the specified MongoDB database and collection that matches the passed-in data. The number of documents deleted is returned to the user. If no documents are deleted, zero is returned.

### Screenshots

The following pages of this README file contain screenshots that have been provided to demonstrate how the Python module functions:

Upload the Austin Animal Center (AAC) Outcomes dataset into MongoDB by importing a CSV file using the appropriate MongoDB import tool. Use the database name “AAC” and collection name “animals.”:

A screenshot of a computer

Description automatically generated

Screenshot of the import command prior to its execution.

1. Screenshot of the import command after its execution:

A screenshot of a computer

Description automatically generated

1. Create a new user called “aacuser” for the database AAC in the mongo shell):

A screenshot of a computer

Description automatically generated

Take a screenshot of your login process to MongoDB using the Mongo shell. Be sure you can access MongoDB and list the databases using both the admin and aacuser accounts:

A screenshot of a computer screen

Description automatically generated

When the Linux terminal is first open, the user is already logged in as the root or administrative user. The db.runCommand({connectionStatus:1}) shows the authenticated user is the root user and is able to see all databases when the “show dbs” command is executed.

The following screenshot for the user: aacuser:

A screenshot of a computer screen

Description automatically generated

This screenshot shows the username and password being changed from the root user to the aacuser. Once in the mongo shell, the db.runCommand({connectionStatus:1}) is again used to show what user is logged into the mongo shell. Lastly, the “show dbs” command is executed to show that the aacuser can only read and write the AAC database.

A screenshot of a computer

Description automatically generated

Import of AnimalShelter class and object instantiation.

A screenshot of a computer

Description automatically generated

Creating a Python dictionary that will be used as a document that we want to create using the

Python module.

A screenshot of a computer

Description automatically generated

Using the create method from the Python module, the document is added to the specified

MongoDB database and collection. True is returned to show the successful insertion of the

document.

A screenshot of a computer

Description automatically generated

When trying to add the same document again, False is returned.

A screenshot of a computer

Description automatically generated

An exception is raised if we try to create a document using a None data type.

A screenshot of a computer

Description automatically generated

Execution of the read method from the AnimalShelter class in the python module. If the data is found it is assigned to query and a for loop is used to iterate through query and display the data to the user.

A screenshot of a computer

Description automatically generated

When a query is made that does not match any existing records, an empty list is returned.

A screenshot of a computer

Description automatically generated

An exception is raised if we try to pass in a None data type to the read method in the Python module.

A screenshot of a computer

Description automatically generated

Execution of the update method from the AnimalShelter class. Two dictionaries are passed in. The first parameter is the search data, and the second parameter is the update data. If any documents are found using the search data, those documents are updated with the update data in the specified MongoDB database and collection. The number of updated documents is returned to the user.

A screenshot of a computer

Description automatically generated

Execution of the same command with the same search and update data results in the number zero being returned because there are no longer any documents in the specified MongoDB database and collection that match the passed in search data.

A screenshot of a computer

Description automatically generated

An exception is raised if we try to pass in a None data type to the update method in the Python module.

A screenshot of a computer

Description automatically generated

Execution of the delete method from the AnimalShelter class. If any documents match the passed in delete data, they are deleted from the specified MongoDB database and collection. The number of documents deleted is returned to the user.

A screenshot of a computer

Description automatically generated

A second execution of the delete method from the AnimalShelter class shows that if no documents are found that match the passed in delete data, no documents are deleted, and the number zero is returned to the user.

A screenshot of a computer

Description automatically generated

An exception is raised if a None data type is passed into the delete method in the Python module.

## Contact

Justin Starr

justin.starr@snhu.edu