# 7-2 Project Two – Tristan Lacey

## About the Project/Project Title

The purpose of this project is to learn more about full stack development, by creating a MongoDB back end and Python front end to create a fully realized application for Grazioso Salvare. The full application will be built with the completion of Project Two. In this project, we focused on building on our Python driver code and completed adding all CRUD operations, which allows a user to manipulate the MongoDB back end. We’ve also set up the database with some data and added user authentication as well.

## Motivation

The client we will be working for on these projects needs a software application that can help to manage their existing data from animal shelters. The application will help them to run their business and is an innovative product being built for an international rescue animal training company called Grazioso Salvare.

## Getting Started

To get started you will need to install MongoDB and set up your database to interact with. A Python install will also be required to utilize the scripts that allow interaction with the database through an application. The database we are working with in this project is the ‘aac’ database and the ‘animals’ collections. Authentication has been setup to allow the user to connect as the ‘aacuser’ through the application, which provides security and limits the permissions of the user. Once connected, you can use the functions provided in the ‘CRUD\_Python.py’ to perform CRUD operations on the database. The MongoDB database will be considered the backend component. MongoDB was chosen because it is a flexible, noSQL database engine that can be used with a number of programming languages. Python is a great choice because it is also flexible and agile, and has support already built to integrate with both MongoDB and the Dash framework.

Next, creation of the Python class (e.x. “Animal Shelter” class in the CRUD\_Python.py file) will be required to allow manipulation of the database through Python/PyMongo as a driver. This will be linked up with the Python/Dash front end component that will allow a user to interface with the application. This file includes code for all of the basic CRUD operations that can be performed on the database, as well as a few more advanced queries used for specific filtering.

Since the latest update, we started using the Dash framework for Python, which allows us to create a web application that interfaces with the database we created. Dash helps us to implemented the MVC model (Model-view-controller), and helps to allow control through “callbacks” / view through a web application “dashboard”. Dash resources have been added under the installation section which will assist in getting started. Using this framework, we are able to add html/css elements to our web application, which creates a dashboard for the client. Currently the code implements an interactive data table, filter options, geolocation through Leaflet, and a pie chart. More can be added to enhance the dashboard, building on the code provided. The resources below will help with installation and setup of the Mongo database, Python glue middleware layer, and the front-end Python/Dash web application which constitute the full scope of this project.

One of the biggest issues I ran into was the port being numbered incorrectly in the constructor. Be careful to make sure the port is correct, which can be identified when running ‘mongosh’ to connect to the MongoDB from a terminal session. Some of the final challenges I ran into were around logic to implement the filter buttons correctly, and ensuring they behaved optimally.

## Installation

In order to use this project, you will need to install/set up the following components. There are also links to documentation to help learn how to use the different components used in this project.

* [MongoDB install](https://www.mongodb.com/docs/manual/tutorial/getting-started/) (non-relational database engine, requires a MongoDB database)
* [Python 3.9.x](https://www.python.org/downloads/)
* [PyMongo](https://pymongo.readthedocs.io/en/stable/) (Python library to interact with MongoDB)
* [PyMongo FAQ](https://api.mongodb.com/python/2.3/faq.html)
* [PyMongo: Database Profiler](https://docs.mongodb.com/manual/tutorial/manage-the-database-profiler/#DatabaseProfiler-EnablingProfiling)
* CRUD\_Python.py (Module to perform CRUD operations)
* MongoClient (Module to support Python and MongoClient integration)
* Dash:
  + [Dash Core Components](https://dash.plotly.com/dash-core-components)
  + [Dash DataTable](https://dash.plotly.com/datatable)
  + [Dash Leaflet Documentation](https://dash-leaflet-docs.onrender.com/)
  + [Pie Charts (Plotly)](https://plotly.com/python/pie-charts/)

## Usage

The Python driver being built for the project will allow the user to perform all of the basic CRUD operations on the MongoDB database. Example of these CRUD operations and testing validation below:

### Code Examples:

The code below is the CRUD operations Python driver. The class can be used to make the connection to the database as the *'aacuser'* user. It also has functions for each of the CRUD operations:

**Database Commands (Mongo Shell and Screenshots): MongoImport and Authentication:**

**MongoImport:**

A screenshot of a computer

Description automatically generated

**Ensure Authentication:**

**A screenshot of a computer

Description automatically generated**

**CRUD\_Python.py:**

from pymongo import MongoClient

from bson.objectid import ObjectId

***#Class to handle connection to MongoDB with fixed connection information in the constructor***

***#Class will also handle CRUD operations in MongoDB***

*class AnimalShelter(object):*

*""" CRUD operations for Animal collection in MongoDB """*

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

*# Initializing the MongoClient. This helps to*

*# access the MongoDB databases and collections.*

*# This is hard-wired to use the aac database, the*

*# animals collection, and the aac user.*

*# Definitions of the connection string variables are*

*# unique to the individual Apporto environment.*

*#*

*# You must edit the connection variables below to reflect*

*# your own instance of MongoDB!*

*#*

*# Connection Variables*

*#*

*USER = 'aacuser'*

*PASS = 'SNHU1234'*

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

*PORT = 30313 #fixed port*

*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)]*

***# 'CREATE' function to insert documents into MongoDB (C)RUD***

*def create(self, data):*

*#If data is passed successfully, try to add it to the database*

*if data:*

*#Try for exception handling*

*try:*

*print("Starting import...")*

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

*print("Success!")*

*return True*

*#Catch for exception handling*

*except Exception as e:*

*print('Something went wrong: '+ str(e))*

*return False*

*else:*

*print("Nothing to create, because data parameter is empty")*

*return False*

***# 'READ' function to retrieve & read document data C(R)UD***

*def read(self, search):*

*if search:*

*#Try for exception handling*

*try:*

*#Take search results and create a new list from it by appending each search result*

*result = self.database.animals.find(search)*

*resultList = []*

*for cursor in result:*

*resultList.append(cursor)*

*#Return the new list for the user, will return an empty list if no search results found*

*return resultList*

*#Catch for exception handling*

*except Exception as e:*

*print('Something went wrong: '+ str(e))*

*return exception*

*else:*

*print("Search parameters were empty, please try again.")*

***# 'UPDATE ONE' function to update an existing document CR(U)D - Uses update\_one MongoDB method***

*def updateOne(self, targetDoc, dataToUpdate):*

*if targetDoc and dataToUpdate:*

*#Try for exception handling*

*try:*

*result = self.database.animals.update\_one(targetDoc, dataToUpdate)*

*totalUpdates = result.modified\_count*

*if totalUpdates > 0:*

*print("The following number of documents were updated successfully:")*

*return totalUpdates*

*else:*

*print("Failed to update, number of documents updated:")*

*return totalUpdates*

*#Catch for exception handling*

*except Exception as e:*

*print('Something went wrong: '+ str(e))*

*return exception*

***# 'UPDATE MANY' function to update an existing document CR(U)D - Uses update\_many MongoDB method***

*def updateMany(self, targetDocs, dataToUpdate):*

*if targetDocs and dataToUpdate:*

*#Try for exception handling*

*try:*

*result = self.database.animals.update\_many(targetDocs, dataToUpdate)*

*totalUpdates = result.modified\_count*

*if totalUpdates > 0:*

*print("The following number of documents were updated successfully:")*

*return totalUpdates*

*else:*

*print("Failed to update, number of documents updated:")*

*return totalUpdates*

*#Catch for exception handling*

*except Exception as e:*

*print('Something went wrong: '+ str(e))*

*return exception*

***# 'DELETE ONE' function to delete an existing document CRU(D) - Uses delete\_one MongoDB method***

*def deleteOne(self, docToDelete):*

*if docToDelete:*

*#Try for exception handling*

*try:*

*result = self.database.animals.delete\_one(docToDelete)*

*totalDeletions = result.deleted\_count*

*if totalDeletions > 0:*

*print("The following number of documents were deleted successfully:")*

*return totalDeletions*

*else:*

*print("Failed to delete, number of documents deleted:")*

*return totalDeletions*

*#Catch for exception handling*

*except Exception as e:*

*print('Something went wrong: '+ str(e))*

*return exception*

***# 'DELETE MANY' function to delete an existing document CRU(D) - Uses delete\_many MongoDB method***

*def deleteMany(self, docsToDelete):*

*if docsToDelete:*

*#Try for exception handling*

*try:*

*result = self.database.animals.delete\_many(docsToDelete)*

*totalDeletions = result.deleted\_count*

*if totalDeletions > 0:*

*print("The following number of documents were deleted successfully:")*

*return totalDeletions*

*else:*

*print("Failed to delete, number of documents deleted:")*

*return totalDeletions*

*#Catch for exception handling*

*except Exception as e:*

*print('Something went wrong: '+ str(e))*

*return exception*

### Tests/Screenshots:

The screenshot below shows how the CRUD methods can be utilized to manipulate the MongoDB database:

### Create/Read:

A screenshot of a computer

Description automatically generated

### Read Example 2:

A screenshot of a computer program

Description automatically generated

***NOTE\*\*: For the Update and Delete examples, five copies of Tiki were created (Tiki1, Tiki2, Tiki3, Tiki4, and Tiki5) to use as an example, in addition to the existing Tiki. Queries were run in sequential order below:***

### Update One:

A screenshot of a computer program

Description automatically generated

### Update Many:

A screenshot of a computer

Description automatically generated

### Delete One:

A screenshot of a computer

Description automatically generated

### Delete Many:

A screenshot of a computer

Description automatically generated

### Dashboard Usage:

The UI/UX of the dashboard used in this project was created using Python and Dash. The data in the dashboard can be manipulated by using filters. There are three preset filters: “Water Rescue”, “Mountain Rescue”, and “Disaster Rescue”. The data can also be manually filtered by using the filters at the top of each column, or by sorting columns by clicking the column title. Lastly, the “Reset Filter” will revert the dashboard back to an unfiltered state.

### Unfiltered Dashboard Example:

A screenshot of a computer

Description automatically generated

### Water Rescue Filter Example:

A screenshot of a computer

Description automatically generated

### Mountain Rescue Filter Example:

A screenshot of a computer

Description automatically generated

### Disaster Rescue Filter Example:

A screenshot of a computer

Description automatically generated

## Roadmap/Features (Optional)

Next future release will add additional UI/UX elements, additional analytics, and any other additional requirements needed by the client Grazioso Salvare. Personally, I would also like to optimize some of the code and continue to improve on the underlying Python driving the project dashboard.

## Contact

Your name: Tristan Lacey