# CS 340 Grazioso Salvare README

## About the Project/Project Title

This project was created to provide CRUD functionality for an Anima Shleter database hosted on MongoDB using an API designed by the Dash Framework in Python. The Python modules are designed to be imported the Dash framework, and then used to create, read, update, and delete files in MongoDB. The project also requires user authentication for secure database management.

Once the database is accessed, the project requires a user interface capable of searching and selecting animals of the required condition. Query based on specified requirements are expected to return appropriate list results.

## Motivation

This project exists to create a seamless interface for users to perform CRUD operations and maintain their Mongo database. Most applications will need a secondary program to allow users to maintain database files on their systems, without having to interact directly with Mongo. This project exists for developers to have a Python module they can install in their system to allow ease of access into the Mongo system.

Once accessed, this project aims to have a fluid and well-designed user interface capable of handling user input. The user should be able to query the database without having to utilize mongo or python functionality.

MongoDB is the model component for this development as it can interact with the built in Python module, and output to the Dash Interface. The JSON style dictionary entries are easily queried, and output a datatype that can be easily read by the user.

The Dash framework provides the ability to output different apps for the user. These apps allow the user to query data in a way that in easily read and sensible.

## Getting Started

To create this module on your system you first need to have access to MongoDB and a usable database. You can import databases directly in MongoDB using the built in mongoimport tool (see screenshot at end of README.

Database access should be limited based on user needs. You can create user authentication directly in MongoDB using the built in db.createUser() function.

Once the MongoDB database and users are set up, create a module in Python with a call to this database and provided authentication. This will give your module the ability to perform CRUD operations. Each method within the class should perform a single CRUD operation for simplicity. Pass any data through to each method as a dictionary, and then use the provided python dot notation methods to perform CRUD operations on the database.

Following creation of the Mongo and Python modules, create a user interface using the Dash framework. This framework will operate suing HTML, with the ability to call the Python and Mongo Modules. In the Dash framework each “app” will display independently, and function based on the queries to the Mongo database.

## Installation

The tools needed to use this software are the following imports:

pymongo from MongoClient

bson.objectid from ObjectId

pprint from pprint

dash\_html\_components as html

PyMongo is the Python driver provided by MongoDB to allow Mongo functionality within Python. This driver will need to be installed to use the software. PyMongo was chosen as it officially supported by MongDB. CRUD functions come preinstalled making it an easy transition.

ObjectId is necessary to create a hexadecimal id for any new entry into the database. Without this import the objects will not load correctly into the Mongo database.

Pprint or Pretty Print helps the interpreter handle the data from on system to another. It helps to format all the data in a way where nothing is lost to the interpreter.

dash\_html\_components is the Dash interface library which helps to display the final results. Using HTML language, Dash takes input from Python, which can access the Mongo database, to output the information to the user in a seamless fashion.

## Usage

### Code Example

The purpose of this library is to perform CRUD operations on a Mongo database using Python.

The module will have methods for each CRUD function, as well as user authentication.

Create: The create function exists to create new entries into the database. The command insert\_one() is used. This function will create a new instance of the provided dictionary item with a unique hexadecimal id number.

Read: The read function exists to find and output any queried database entries. The command find() is used. This function will scan the database for any entries that match the query, and then output them in a list.

Update: The update function exists to change a designated key:value pair for an entry in the database. The command update\_many() is used. This function will find any instance in the dictionary with a specified key:value pair, and update the secondary provided key:value pair.

Delete: The delete function exists to delete a chosen entry. The command delete\_one() is used. This function will find the desired key:value pair and delete the entry from the database.

class DatabaseCrud(object):

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

# Initializing the MongoClient. This helps to access the MongoDB databases and collections.

# This step will also include user authentication.

self.client = MongoClient('mongodb://%s:%s@localhost:xxxx/database' % (username, password))

self.database = self.client[‘database’]

def create(self, data):

self.database.collection.insert\_one(data)

def read(self, data):

list(self.database.collection.find(data))

def update(self, data, update\_data):

self.database.collection.update\_many(data, {"$set": update\_data})

def delete(self, data):

self.database.collection.delete\_one(data)

Create a clickable sorting option that will show animals based on the company specified requirements.

dcc.RadioItems(

id='filter-type',

#Create filter options based on copmany specifications

options=[

{'label': 'Water Rescue', 'value': 'WR'},

{'label': 'Mountain/Wilderness Rescue', 'value': 'MWR'},

{'label': 'Disaster Rescue/Individual Tracking', 'value': 'DRIT'},

{'label': 'Reset - returns unfiltered state', 'value': 'RESET'}

],

value='RESET',

labelStyle={'display': 'inline-block'}

Display an unfiltered list of the entire database.

dt.DataTable(

id='datatable-id',

columns=[

{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns

],

#Create basic data table

data=df.to\_dict('records'),

### Tests

This code can be tested by attempting to create, read, update, and delete an entry using the module.

Import the module into python, and create an instance of the class. Then add a document and attempt to read, update, and delete it.

from aac\_proj1 import AnimalShelter

aac\_database = AnimalShelter()

aac\_database.create({"age\_upon\_outcome" : "Test", "animal\_id" : "123456", "animal\_type" : Cat", "breed" : "Test", "color" : "Test", "date\_of\_birth" : "2022-09-30", “datetime" : "22022-09-30 12:00:00", "monthyear" : "2022-09-30T12:12:12", "name" : "", “outcome\_subtype" : "Test", "outcome\_type" : "Test", "sex\_upon\_outcome" : "Test", “location\_lat" : 30.6525984560228, "location\_long" : -97.7419963476444, “age\_upon\_outcome\_in\_weeks" : 52.9215277777778})

aac\_database.read({"age\_upon\_outcome" : "Test"})

aac\_database.update({"age\_upon\_outcome" : "Test"}, {"animal\_type" : "Test"})

aac\_database.delete({"age\_upon\_outcome" : "Test"})

### Screenshots

**Text

Description automatically generated**

*Text

Description automatically generated*

*Text

Description automatically generated*

Next, run the dashboard to ensure the proper filtering.

A picture containing table

Description automatically generatedA picture containing graphical user interface

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface, application

Description automatically generated

## Contact

Noah Coleman