# CS 340 README for Custom Pymongo CRUD Library

## About the Project/Project Title

This class facilitates Create-Read-Update-Delete (CRUD) actions in MongoDB via the Pymongo driver. The class will contain all of the abstracted getter/setter methods needed for CRUD operations and MongoDB initiation.

## Motivation

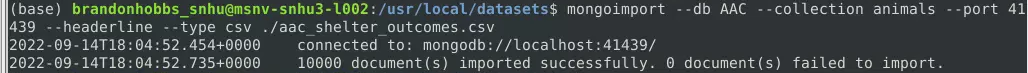
CRUD operations are commonplace when using any database but formatting these queries and any error handling can be tedious. To help alleviate the tedium of formatting each new query or re-code for each error type these operations have been abstracted into simple *get* and *set* styled methods.

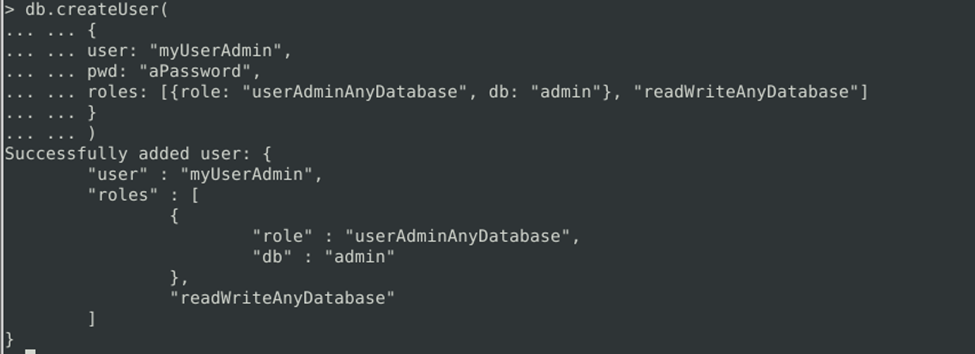
## Getting Started

To start using this abstraction class the following simple guide is provided for your reference.

## Installation/Pre-Conditions

To use this abstraction the following tools must be installed:

* Python 3.6
* Pymongo 4.2
* MongoDB 4.2
* Data available in a Mongo database collection, see example on importing a CSV  
  
* User account with permissions, see example on creating an admin role, user role and their authentication (you may choose permissions other than shown)  
  Text

  Description automatically generated  
    
    
    
  Text

  Description automatically generated

## Usage

To use this abstraction the following methods are supplied:

* **The following properties are available within the class to help understand any recent interactions**
  + records\_updated
  + records\_matched
  + records\_deleted
* **Constructor (password, username)**  
    
  The constructor will take care of initializing the MongoDB server, connecting to the database.
* **createRecord (data)**  
    
  **createRecord** will create the record with the data in the argument. The argument must be formatted according to the Pymongo API. See below for an example. **True** will be return if the Create action was successful. **False** will be returned if the action was unsuccessful.  
    
  If no argument is passed an exception will be raised and an error message will be passed to the user.
* **deleteRecord (query)**deleteRecord will search the database for items matching the query. The items found will be deleted without warning. **True** will be returned if the number of records is greater than 0, otherwise **False** is returned. The number of records deleted will be stored in the property, **records\_deleted**.  
    
  If no argument is passed an exception will be raised and an error message will be passed to the user.
* **getRecordCriteria (criteria = None)**  
    
  If a record or records are to be located using criteria this method should be used. The criteria may be one or many. The argument must be formatted according to the Pymongo API. See below for an example.  
    
  By default the criteria is None while will return all of the records.
* **getRecordId (data)**  
    
  If a record needs to be located by the server assigned GUID this method may be used.
* **updateRecord (query, newValue)**  
    
  **updateRecord** will search the database for documents matching the query. Subsequently, the documents values matching the key:value pairs in **newValue** will be written in place of the existing data. **True** will be returned if the number of records modified is greater than 0, otherwise **False** is returned. The number of records updated will be stored in the property **records\_modified**. The number of records matched will be stored in the property **records\_matched**.  
    
  If no arguments are passed an exception will be raised and an error message will be passed to the user.

### Code Example

Example to create the following record:  
  
*animals.createRecord({*

*'age\_upon\_outcome': "1 year",*

*'animal\_id': 'test\_id',*

*'animal\_type': 'test',*

*'breed': 'test breed',*

*'color': 'color',*

*'date\_of\_birth': '1900-01-01',*

*'datetime': '1900-01-01 12:00:00',*

*'monthyear': '1900-01-24T12:00:00',*

*'name': 'name',*

*'outcome\_subtype': '',*

*'outcome\_type': 'test',*

*'sex\_upon\_outcome': 'test',*

*'location\_lat': 10.10,*

*'location\_long': -10.10,*

*'age\_upon\_outcome\_in\_weeks': 123.123*

*})*

*)***True** was returned as the record was successfully written to the database.  
  
Example to query for a dog named Rex with object *animal*:

1. **Create the object animal and initialize the MondoDB server via the Constructor**  
     
   *animal = AnimalShelter(‘password’, ‘username’)*
2. **Create the query object that will store the returned documents**  
     
   *query = animals.getRecordCriteria({"name": "Rex"})*
3. **If the age of *Rex* should be *2 months* then use this query**  
     
   *query = animals.getRecordCriteria({"name": "Rex", 'age\_upon\_outcome': '2 months'})*

### Tests

Code examples from previous examples:

Text, letter

Description automatically generated

A picture containing timeline

Description automatically generated

## Example for deleting a record

## The number of documents deleted can be determine with the available property

## *Output:*

## 

## Example for updating a record with the current datetime string

## The details of the matched and modified documents may be determined with the appropriate properties

## 

## *Output:*

## 

## Errors will be returned to the user’s console for debugging purposes. Here is an improperly formatted deleteRecord()

## 

## *Output:*

## Roadmap/Features (Optional)

Future development will focus on MongoDB initialization. Currently, the database must be hardcoded. In future releases this will be able to be passed to the constructor.  
  
Also, the constructor will try and initialize the database for each new object. A singleton will be added in future releases.

## Contact

Your name: Brandon Hobbs, Created for SNHU CS-340 om September 28, 2022