# CS 340 README Template

## About the Project/Project Title

AnimalShelter – a Python library for interacting with an Animal Shelter MongoDB database

## Motivation

The goal of AnimalShelter is to make it simple to create, find, update, and delete animal shelter records using Python.

## Getting Started

Using this library is easy.

1. Download the aac\_crud\_driver.py.
2. Import the module and create an AnimalShelter object with server config details:

* from aac\_crud\_driver import AnimalShelter  
    
  config\_dict = { "username": "aacuser", "password": "123456" }  
    
  shelter = AnimalShelter(config\_dict)  
    
  lucy = shelter.find( { "name": "Lucy", "breed": "Dog" } )

## Installation

This library requires **Python 3** and the **pymongo** library, which is used as the backend MongoDB driver.

1. **Install Python 3.x**
   * Download and install it from [https://www.python.com/](./the%20Python%20official%20website)
2. **Install MongoDB**
   * Follow MongoDB’s [https://www.mongodb.com/docs/manual/installation/](./official%20guide)
3. **Install pymongo**
   * pip install pymongo or apt install python3-pymongo (or the package manager of your choice)
4. **Clone the repo**
   * git clone https://github.com/aldonadi/TODO-UPLOAD-REPO

## Usage

### Server Connection and Authentication

The main Python file aac\_crud\_driver.py contains some (hopefully) reasonable default values for MongoDB server hostname, port, username, etc. It assumes a locally-hosted MongoDB at the default port. To reconfigure this and supply your own credentials, there are two options: you can specify them in a dict you pass to the constructor, or put them in a db.yml file that is in the same directory as aac\_crud\_driver.py.

#### Option 1: Specifying in the constructor

shelter = AnimalShelter(   
 {   
 "hostname": "db.example.com",   
 "port": 12345,  
 "username": "user1",  
 "password": "very-insecure1",  
 "db\_name": "main-street-shelter",  
 "collection\_name": "animal\_records"  
 } )

#### Option 2: Specifying in db.yml

In db.yml:

hostname: "db.example.com"   
port: 1234  
username: "user1"  
password: "very-insecure1"  
db\_name: "main-street-shelter"  
collection\_name: "animal\_records"

In your Python script:

shelter = AnimalShelter()

### Code Example

from aac\_crud\_driver import AnimalShelter  
# create the driver object  
driver = AnimalShelter()  
  
# add a new record into the database using regular Python dicts  
driver.create( { "breed": "Wombat", "Name": "Spunky" } )  
  
# get a list of matching records  
matches = driver.find( { "breed": "Zebra" } )

## Functions

* *create(data)*: Takes a dictionary object representation of the record to add. It returns True if the insertion succeeded and False if it did not.
* **Example:**
* if shelter.create( { "breed": "Wombat", "name": "Huggles", "age": 5 } ):  
   print("A new wombat has joined our shelter")
* *find(query [, include\_id=False])*: Takes a dictionary object representing search criteria. Returns a list of all matching records. If no records are found, returns an empty list. By default, the Mongo ObjectId of matching records is not included. Set include\_id to True to include ObjectId in returned results.
* **Example:**
* shelter.find(  
   query = { "breed": "Wombat", "name": "Huggles", "age": 5 },  
   include\_id = True)
* *update(query, newdata)*: Updated all documents that match the query dictionary with the fields and values in the newdata dict. Returns a dict in the form
* {   
   "success": bool, # True if at least 1 document was modified  
   "modified\_count": int # Number of documents that were modified  
  }
* Raises an exception if either query or newdata are not dicts.
* **Example:**
* birthday\_boy = { "breed": "Wombat", "name": "Huggles", "age": 5 }  
  new\_age = { "age": 6 }  
    
  if shelter.update(birthday\_boy, new\_age)['success']:  
   print("Happy birthday!")
* *delete(query)*: Deletes all documents that match the query dictionary. Returns a dict in the form:
* {   
   "success": bool, # True if at least 1 document was deleted   
   "deleted\_count": int # Number of documents that were deleted  
  }
* Raises an exception if query is not a dict.
* **Example:**
* adopted\_wombat = { "breed": "Wombat", "name": "Huggles", "age": 6 }  
    
  if shelter.delete(adopted\_wombat)['success']:  
   print("Another happy family!")

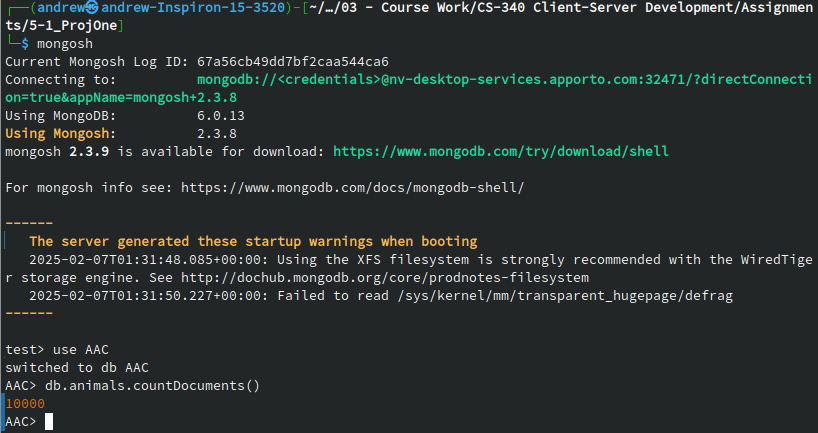
### Testing

To test the functionality, use the following:

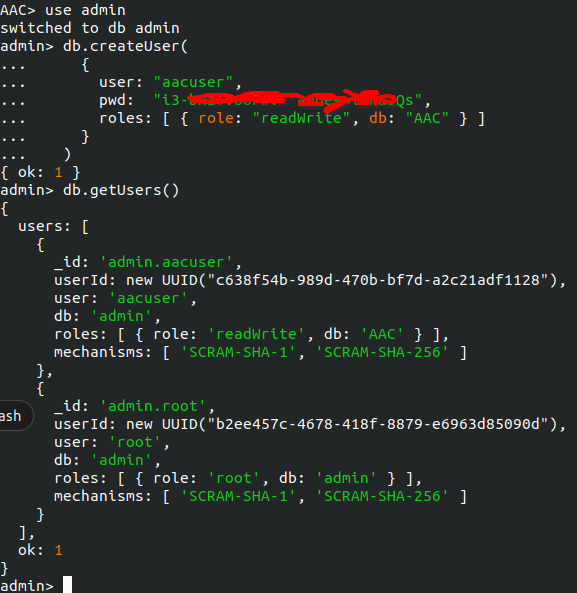
# Test inserting a valid record  
assert driver.create({"breed": "Dog", "name": "Buddy"}) == True  
  
# Test inserting an invalid record (should return False)  
assert driver.create(None) == False  
  
# Test reading a record that exists  
assert len(driver.find({"breed": "Dog"})) > 0  
  
# Test reading a record that doesn't exist (should return empty list)  
assert driver.find({"breed": "Dragon"}) == []  
  
# Test updating a record  
assert driver.update( { "breed": "Dog", "name": "Buddy"}, # query dict  
 { "name": "Bud" } # dict with into to update  
 )['success'] # will be True if at least 1 document was updated  
  
# Test deleting a record  
assert len(driver.find( { "objectId": "67a56b32ec2435f6e169c472" })) == 1 # verify it exists  
assert driver.delete( { "objectId": "67a56b32ec2435f6e169c472" })['success'] # delete it  
assert len(driver.find( { "objectId": "67a56b32ec2435f6e169c472" })) == 0 # verify it is gone

### Screenshots

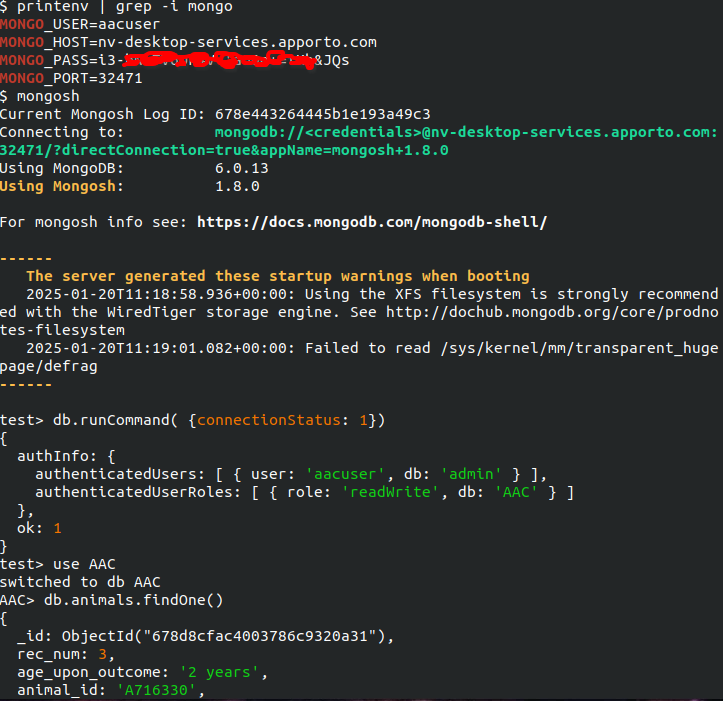
**MongoDB Import Execution**:



**User Authentication Execution**:

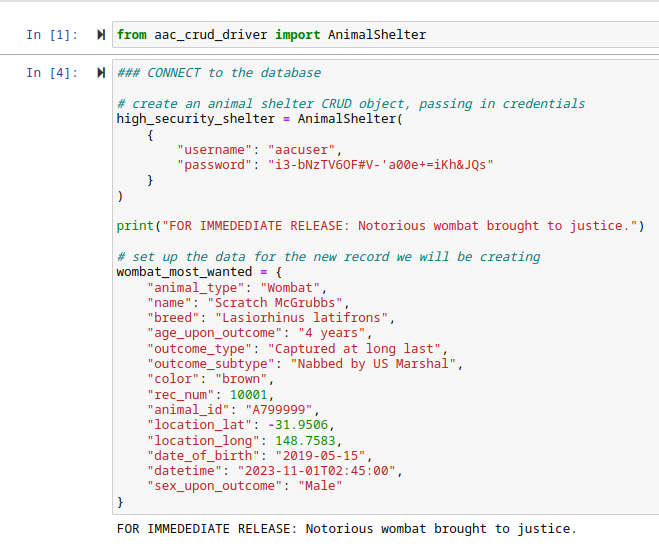


**CRUD Functionality Test Execution**:

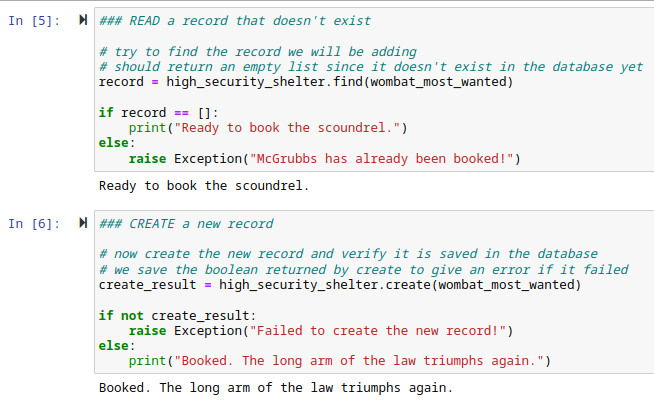


**CRUD Functionality Test Execution**:

*Importing the module, creating the driver object, and setting up a new document dict.   
Scratch McGrubbs is a dastardly scoundrel thankfully locked up for good now.*



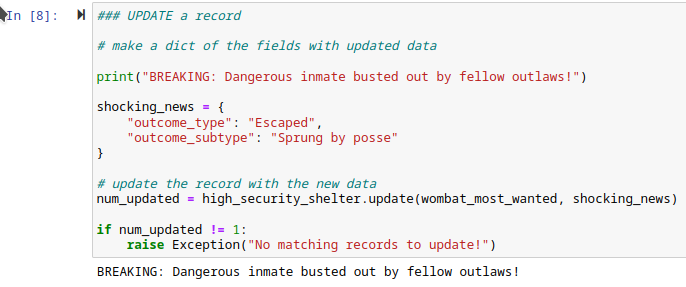
*Verifying that the new record does not exist yet (he isn’t already booked), and then creating it successfully (booking him):*



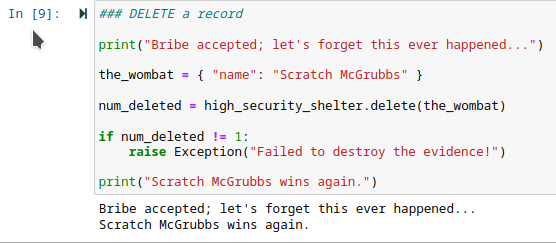
*Proving that he was booked successfully by finding the record we just inserted:*



*Updating a record that already exists. Curses! He’s been sprung loose!*



*Deleting a record. Sometimes a few thousand dollars passed into the right hands lets a criminal live another day.*



## Challenges

Getting the authentication to go through has been the most difficult to get right so far. Credentials need to be url-escaped (which is now handled in the driver code); this was not an easy requirement to figure out.

Refactoring the database connection and authentication config away from the source code and into a YAML file took a bit of work, especially getting a nice order-of-precedence.

## Roadmap/Features (Optional)

* ☒ Implement the **update** and **delete** functions
* ☒ Improve the way database connection/authentication details are stored (e.g. a YAML file)
* ☐ Perform validation on the dict data when creating a new record
* ☒ Update API for better return values from update and delete
* ☒ Hide ObjectID from found documents by default

## Contact

Andrew Wilson