# Christian Rojas | CS 340 README

**About the Project/Project Title**

This project is an application built using a CRUD handler API to manage the Austin Animal Shelter database of animals in MongoDB. This application takes the user’s input and depending on what they need to do, can create data, read entries, update entries, and delete data. The Dash database presents the data to the user in an organized and ingestible form.

## Motivation

The need for the application arose after Austin Animal Shelter needed a better way of managing their database. A CRUD handler is an efficient way to manage MongoDB. By using Python, the application will allow the user to search through their database, create new data, and manage their data by deleting or changing entries.

As for the dashboard, Dash was the chosen tool. Dash is a react.js tool that can be used to build strong and responsive user interfaces. For this dashboard, dash tags are used for html to create a header to include the logo and title. Dash also allowed for a filter-responsive data table and corresponding pie chart.

## Getting Started

To get started, you will need to follow these steps:

* **Part One:** Accessing a database in MongoDB
* **Part Two:**  Create user credentials
* **Part Three:** importing data and testing queries
* **Part Four:** Create a dash web application

**Part 1:**

1. Download the testScript.py and the animal\_shelter.py files. The animal\_shelter.py file contains the CRUD handler, and testScript.py is required to test the CRUD functionality.
2. Next, you need to access the “AAC” database – you can do this by following these instructions:
   1. In your mongo shell:
      1. mongod\_ctl start-noauth (by including “-startnoauth” you can start the mongo and access the database without credentials)
      2. show dbs (This will show you the databases available)

**Part 2:**

1. Next, we need to create an admin account and authenticate it:
   1. Start with the command:
   2. Mongod\_ctl start-noauth
   3. mongo
   4. > db.createUser({

user: "username", pwd: "password",roles: [{

role: "readWrite",

db: "AAC"

}]})

1. This will create an account with your selected username and password. Be sure to give the user the role access to read and write in the database AAC.
2. To log in with the new user, enter the following command into the mongo shell:
   1. mongo –port ##### "admin" -u "admin" -p --authenticationDatabase AAC
3. Once prompted, enter your password.
4. To verify you are logged in use commands:
   1. > mongo
   2. >show dbs (If authentication worked, your databases will show).
   3. >use AAC

**Part 3:**

1. Next step is to take the animal\_shelter.py, and testScript.ipynb files and put them in a folder together. This is important, as the script cannot work if the CRUD handler is not in the same directory. Once in the same folder, load the testScript file into a Jupyter Notebook and run the script.
   1. You can change the scripts variables to fit your needs and test parameters accordingly

**Part 4:**

1. Create a new Dash application, then using a combination of HTML and CSS, configure the dashboard to the desired layout. For this project, create a header, data frame, map, and chart.
   1. Ensure all parts of the dashboard are identified appropriately.
2. Create an app callback to populate the data frame.
3. Create radio buttons and preset the desired queries. For this project, complex queries are required.
4. Create a callback to update the map based on the top menu item of any query.
   1. When a user selects another item, the map should update to the location of the newly selected item.
5. Create the pie chart that will display data based on the selected query. This chart should change depending on the selected filter via an app callback.

## Installation

* *Python 3.10.4:* Python is the language that is used for this application. Downloading the latest version is *highly* recommended.
* *MongoDB v4.2.6:* MongoDB is the database software that can securely host the animal shelter database. MongoDB pairs well with Python to create a fast and secure connection.
* *Pymongo:* A Python library that allows Python to integrate with MongoDB. This is required to ensure the application can reliably connect to the database.
  + Use the following command to install Pymongo:
  + *pip install pymongo*
* *Dash:* Dash is a framework for web applications. This tool can be imported using:
  + Import dash

## Usage

This application is used to display the rescue animals in a data set. The first function is to allow the user to select between 3 preset filters. The data is presented in three ways: a data table, that displays all relevant data to the animal. The second, is a pie chart that displays the percentage of each breed in the selected category, and the third is a map that shows the animals location.

Below are screenshots of the applications different functions:

Reset:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Disaster Rescue:

A picture containing text

Description automatically generated

Graphical user interface, application

Description automatically generated

Mountain Rescue:

A picture containing table

Description automatically generated

Graphical user interface, application

Description automatically generated

Water Rescue:

Table

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Chart

Description automatically generated

Multiple markers:

Map

Description automatically generated

### Code Example

* **Create(self,data=dict()):** This Create method passes python dictionaries along to MongoDB, since Python dictionaries and Mongo databases are structured similarly, Python is the optimal language for a MongoDB CRUD handler. Within the function, *insert\_one* is used to create a new database entry.
* **Note:** using *insert* instead of *insert\_one* or *insert\_many* will result in a deprication warning when testing.
* **Read(self,data):** This Read method takes search criteria from the dictionary and searches for it against the database. If nothing is read, it will throw an exception signaling that no such query was found.
* **Update(self, fromData, toData, count):** The update method allows pre-existing data to be updated by the user. It takes the user’s query and matches it with an entry in the database. Example, a user wants to change the status of a pet. The user can use the pets ID to look up the data and change the status. If the function fails, it will raise an exception, if the function cannot find the matched data, it will let the user know there is an error.
* **Delete(self, data, count):** This Delete method will take the user’s query and search for it in the database, if the query is found, it will delete the matching query. I also created a function to allow the program to delete more than one query at a time. Should the function fail, it will raise an exception and notify the user.
  + For this function, I used *try except* for testing.

### Tests

For my tests, I created test data to run against the CRUD handler via unit testing. Below are some screenshot examples:

Text

Description automatically generated

Text

Description automatically generated

Text, chat or text message

Description automatically generated

### Screenshots

Screenshots of ‘aacuser2’ admin account creation:

Text

Description automatically generated

Screenshot logging in as ‘aacuser’

Text

Description automatically generated

## Contact

Your name: Christian Rojas