# CS 340 README Template

## About the Project/Project Title

This project aims to create a custom software design for a client that intends to search for potential rescue training dogs based on certain criteria that are available through several nonprofit agencies that operate several animal shelters.

Global Rain’s project creates a MongoDB database consisting of existing data from the animal shelters, as well as a client-facing web application dashboard that is used to interact with the database. As part of the process, Python was used to lay down the foundation for CRUD interaction with the database. Specifically, the Creation and Reading of data tied to the existing MongoDB database, as well as the ability to Update and Delete content as well.

Additionally, the client requested a web application dashboard that could provide the means for a user to log into the platform, access the database, and use built-in analytics to review the data as well as interact with the dashboard itself to filter content based on request criteria.

## Motivation

A rescue-animal training company by the name of Grazioso Salvare seeks to further advance their efforts of finding solid candidates for search and rescue training. To optimize the process of identifying candidates for training, Grazioso Salvare partners with several animal shelters.

The main focus and motivation for this portion of the project was to optimize the management of currently existing data across five animal shelters. Reusable and portable Python modules drastically simplify the process of adding new records, updating existing records, and searching through the data. By establishing a MongoDB database to house the data and manage user authentication, as well as utilizing Python’s powerful scripting capabilities, the groundwork for the rest of this project can be established by implementing CRUD functionality. The criteria for good candidates for search and rescue training require the ability to search through a large amount of data in a flexible and easy to access manner.

By creating a dashboard that utilizes the database, the client can easily access the aggregated data from their animal shelter partners and perform any sort of research into the available animals. Considering the fact that there is a specific criteria the client is looking for in dogs depending on the type of rescue needed, the dashboard offers an option to quickly filter, analyze, and display data entries matching what the client is looking for.

## Getting Started

To get started working with this project,

* Begin by cloning the repository containing the animal\_shelter.py and the testScript.ipynb files
* Access Linux shell
* Set up a MongoDB environment with the provided files.
* Load database using the following commands  
  cd /usr/local/datasets

mongoimport --username="${MONGO\_USER}" \

--password="${MONGO\_PASS}" --port=${MONGO\_PORT} \

--host=${MONGO\_HOST} --db enron --collection emails \

--authenticationDatabase admin --drop ./enron.json

* Access mongo shell using ‘mongosh’
* Create a user through the admin database:  
  db.createUser(  
   {  
   user: “aacuser”,  
   pwd: “<insert password here>,  
   roles: [{ role: “readWrite”, db: “aac”  
   }  
  )
* An example of this can be seen here: A screenshot of a computer screen

  Description automatically generated
* Take note of environment details by running:  
  MONGO\_USER=aacuser  
  MONGO\_PASS=<insert password here>  
  printenv | grep -I mongo
* Update animal\_shelter.py file to ensure host, port, etc. match environment details
* From PowerShell, change directory to location where you stored the .py and.ipynb files
* From that directory, run a jupyter notebook:   
  jupyter notebook
* You should have the notebook and Python file available to run.
* With the Jupyter notebook available,

## Installation

* Download and Install Python, preferably at version 3.0 or greater:  
  <https://www.python.org/downloads/>
* Install an IDE for Python, such as PyCharm:  
  https://www.jetbrains.com/help/pycharm/installation-guide.html
* Ensure Pip is installed:  
  <https://www.geeksforgeeks.org/how-to-install-pip-on-windows/>
* Set up MongoDB depending on your system:  
  https://www.mongodb.com/docs/manual/installation/  
  https://www.mongodb.com/try/download/community
* Install Pymongo by running:  
  pip install pymongo
* Install Jupyter Notebook by running:  
  pip install jupyter
* Download the necessary CSV files provided in this repo

## Usage

This Python module was designed to implement CRUD functionality so that a user can easily create or read documents from the MongoDB database. MongoDB was selected as the option for a NoSQL database offering an effective and efficient means to access the large amount of data used in the creation of this dashboard. MongoDB employs a JSON based formatting when storing documents, which allows for effective scaling. Additionally, MongoDB offers an incredibly smooth integration with Python via PyMongo. Since the CRUD functionality of the dashboard relies on Python, MongoDB’s capabilities with PyMongo make it a great choice.

The Dash Framework by Plotly was used to build the actual front-end dashboard available in the web application. This is a commonly used framework that is open-source and effective with handling analytics through its built-in visualization options and streamlined interface. Its compatibility with Python allows for an easy integration when it comes to including the Python CRUD functionality already created. By using the Dash framework, the dashboard has access to modular components, analytic visualizations, and a smooth integration with Python CRUD functionality. For more information on the Dash framework, visit [here](https://dash.plotly.com/installation).

**Code Example**

An example of the Python module can be seen below, where the create and read methods of CRUD are defined and demonstrated. The module checks if an inserted document is empty, and proceed to try inserting it into the MongoDB database “AAC”. If it is unable to, an exception occurs and message is printed. Otherwise, the user is informed of the empty data parameter. Additionally, the read method attempts to assign the found document using the prompt and returns the results as a list. Otherwise, an empty list is returned.

def create(self, doc): # defines create method passing the self and data  
 # If the doc exists, try to insert into collection  
 if doc:  
 # Try to insert into collection and return true if able to  
 try:  
 self.collection.insert\_one(doc)  
 print("Document inserted: ", doc) # Print confirmation of document inserted  
 return True  
  
 # Return false and print error message if exception occurs  
 except Exception as e:  
 print(f"An Error has occurred: {e}")  
 return False  
 # Else, raise ValueError  
 else:  
 raise ValueError("Nothing to save, because data parameter is empty")

Code to accommodate the Update and Delete functionality has been incorporated into the Python module as well. For the Update function, the script will first check if either the query or new values to be used have been left empty, but if they have not, it will try to call on the update\_many Pymongo function while passing the query. In the event that there are not any resulting modified documents, the module will inform the user. If there is any sort of exception or error, the module will raise or print the issue. The delete function behaves very similarly, but instead of updating, it uses the delete\_many function with the passed query.

# Delete method to implement D in CRUD  
 def delete(self, query):  
 if query is not None:  
 # Call delete\_many function with query  
 try:  
 deleted\_docs = self.collection.delete\_many(query)  
 # Check if there were any matches  
 if deleted\_docs.deleted\_count == 0:  
 print(f"No matches found, nothing was deleted.")  
 else:  
 return deleted\_docs.deleted\_count # Return number of deleted documents  
 # Print error message if exception occurs  
 except Exception as e:  
 print(f"An error has occurred: {e}")  
 # Otherwise, if query is empty, print error message  
 else:  
 raise ValueError("Unable to continue, query cannot be empty")

Additionally, examples of the Dash framework code used to incorporate the Python CRUD functionality and MongoDB database can be seen. For example, this section of the code demonstrates the inclusion of radio buttons that enable the user to quickly filter the displayed data by their provided criteria and categories.

html.Hr(),  
html.H4("Filter animals", style={'margin': '0px 0px 2px 10px'}),  
html.Div(className='buttonRow',  
 style={'display' : 'flex', 'padding': '3px'},  
 children=[  
 dcc.RadioItems(['Water Rescue', 'Mountain Rescue', 'Disaster Rescue', 'Reset'], id='filter\_type')]),  
html.Hr(),

The execution of this code and how it was used with the Python CRUD module can be seen here:

def update\_dashboard(filter\_type):  
 # Assign complete database as standard  
 df = pd.DataFrame.from\_records(db.read({}))   
 # If "Water Rescue" button is clicked, use read method to search for matching entries  
 if (filter\_type == "Water Rescue"):  
 df = pd.DataFrame.from\_records(  
 db.read({"animal\_type" : "Dog","breed":{"$in":["Labrador Retriever Mix","Chesapeake Bay Retriever","Newfoundland"]},  
 "sex\_upon\_outcome":"Intact Female","age\_upon\_outcome\_in\_weeks":{"$gte":26},"age\_upon\_outcome\_in\_weeks":{"$lte":156}})  
 )  
 # If "Mountain Rescue" button is clicked, use read method to search for matching entries  
 elif (filter\_type == "Mountain Rescue"):  
 df = pd.DataFrame.from\_records(  
 db.read({"animal\_type" : "Dog", "breed":{"$in":["German Shepard","Alaskan Malamute","Old English Sheepdog", "Siberian Husky","Rottweiler"]},  
 "sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks": {"$gte":26},"age\_upon\_outcome\_in\_weeks":{"$lte":156}})  
 )   
 # If "Disaster Rescue" button is clicked, use read method to search for matching entries  
 elif (filter\_type == "Disaster Rescue"):  
 df = pd.DataFrame.from\_records(  
 db.read({"animal\_type" : "Dog","breed":{"$in":["Doberman Pinscher","German Shepard","Golden Retriever","Bloodhound","Rottweiler"]},  
 "sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":20},"age\_upon\_outcome\_in\_weeks":{"$lte":300}})  
 )  
 # Default option to enable user to backtrack to complete database if needed after using it  
 elif (filter\_type == "Reset"):  
 df = pd.DataFrame.from\_records(db.read({}))   
   
 df.drop(columns=['\_id'], inplace=True)  
 return df.to\_dict('records')

This code allows the user to easily interact with the dashboard. Additionally, code was included to ensure the client has visual analytics, such as an interactive geolocation map and a pie chart that displays the type of breed shown in the current database page.

def update\_graphs(viewData):  
 if viewData is None:  
 return;  
 df2 = pd.DataFrame.from\_dict(viewData)  
 # Set captions and type of graph  
 figure = px.pie(df2, names='breed', title='Animal Breeds')  
 # Prevent overflow   
 figure.update\_traces(textposition='inside')  
 figure.update\_layout(uniformtext\_minsize=12, uniformtext\_mode='hide')  
 return dcc.Graph(figure = figure)

### Tests

A Jupyter Notebook was utilized to test the Python module. Perform an initial test to connect to the database:  
from animal\_shelter import AnimalShelter

CRUD = AnimalShelter("aacuser", "<insert password>")

You can then test the insertion of a new document by creating a new one, such as the following:

testAnimalTwo = {

'age\_upon\_outcome': '4 years',

'animal\_id': 'DN12345',

'animal\_type': 'Jackalope',

'breed': 'Mythical',

'color': 'auburn',

'date\_of\_birth': 'unknown',

'name': 'Jess A. Lupe',

'outcome\_subtype': 'SCRP',

'outcome\_type': 'Transfer',

'sex\_upon\_outcome': 'Neutered Female',

'location\_lat': '30.652343223432',

'location\_long': '-56.534534873899483',

'age\_upon\_outcome\_in\_weeks': '104'}

result = CRUD.create(testAnimalTwo)

print(testAnimalTwo)

Then test the Read method by assigning a prompt to the prompt or query variable and call on the read method. Then iterate through the results to print each document that was returned:

prompt = {'animal\_type': 'Jackalope', 'color': 'auburn'}

results = CRUD.read(prompt)

for document in results:

print(document)

To test the functionality of the update or delete methods, start by first creating a query with the target key, such as ‘animal\_id’ and it’s value. Create a new variable that has the updated values assigned to it, and lastly assigned a new variable to the results of the .update function:  
query = {'animal\_id': 'DN12345'}

newVals = {"color":"Coral"}

updatedDocs = CRUD.update(query, newVals)

print("Documents updated: ", updatedDocs)

A similar process can be completed for the delete method. The reason “delete\_many” was selected is that it can target and delete a single match, but it can also delete however many matches there are as well. In other words, the user is not limited to deleting one entry at a time.   
delete\_query = {'animal\_id' : 'B432159'}

deleted\_docs = CRUD.delete(delete\_query)

print("Documents deleted: ", deleted\_docs)

Testing was also conducted to ensure the features of the dashboard worked as intended. Testing for this included interacting with various features, pages, filter options, visual analytics, and through the use of PyMongo queries, such as this:  
if (filter\_type == "Water Rescue"):

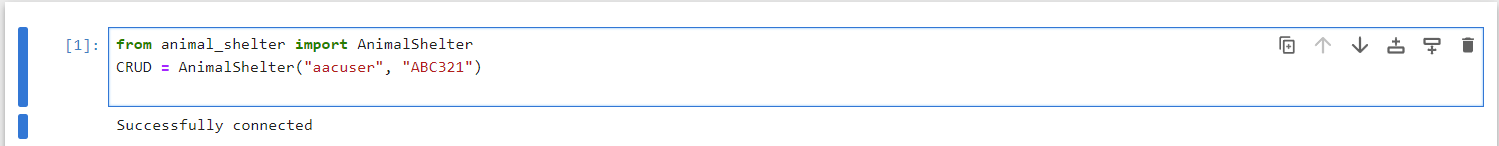
df = pd.DataFrame.from\_records(

db.read({"animal\_type" : "Dog","breed":{"$in":["Labrador Retriever Mix","Chesapeake Bay Retriever","Newfoundland"]},

"sex\_upon\_outcome":"Intact Female","age\_upon\_outcome\_in\_weeks":{"$gte":26},"age\_upon\_outcome\_in\_weeks":{"$lte":156}})

)

### Screenshots

Connecting to database:   


Inserting new document:  
A white rectangular object with a blurry border

Description automatically generated with medium confidence

Testing read method:  
A screenshot of a computer

Description automatically generated

Testing Update method  
  
 Testing Delete Method:  


Testing initial state of the dashboard:  
A screenshot of a computer

Description automatically generated

## Filtering by Water Rescue: A screenshot of a computer Description automatically generated

Filtering by Mountain Rescue:  
A screenshot of a computer

Description automatically generated

Filtering by Disaster Rescue:  
A screenshot of a map

Description automatically generated

Filtering by “Reset”:  
A screenshot of a computer

Description automatically generated

**Steps Taken:**

The steps taken throughout this project began by creating the initial MongoDB database that would house the data used for the rest of the project. For example, the MongoDB database was setup through the steps mentioned in “Getting Started” that involve creating and important the necessary CSV files into the “AAC” database.

Next, a user was created, as mentioned in the steps listed above. The user was given specific permissions for this particular database to ensure the client could incorporate user authentication, if they so choose to. Once the user had been created and the database itself existed, it was time to create the CRUD functionality that would handle how the user can interact with the backend data housed by MongoDB.

Much of the CRUD functionality has already been detailed with steps and screenshots, but he basic CRUD operations were created using Python and PyMongo. This allowed the PyMongo to interact with the MongoDB database, which would eventually perform the various tasks such as reading, writing, updating, and deleting the entries.

With the basic CRUD functionality completed, it was important to then create a platform from which the user could intuitively and easily interact with the data. This was completed through frontend development, which resulted in a web application dashboard using the Dash Plotly framework that ran on PyMongo CRUD operations connected to the MongoDB database. The Python CRUD module allowed the dashboard to perform various MongoDB actions as needed. For example, the interactivity of the dashboard included radio button from which the user could click and filter the animals shown based on their provided criteria. The criteria was incorporated into the dashboard with a combination of PyMongo and the Python CRUD module, which pulled the matching animals and displayed them into the Dashboard.

Additional styling was completed, which incorporated the client’s dashboard with hyperlinked to their website, an individual marker, and general formatting to ensure the dashboard was intuitive and easy to use. Interactive charts were incorporated into the design as well, such as a geo-location map and a pie chart, both of which would automatically update as the user selected various filters or interacted with the dashboard in different ways.

**Challenges and Solutions:**

The biggest challenge I encountered was just being introduced to tools or frameworks I had no experience with whatsoever. It can be a very overwhelming experience to be tossed into something you have never heard of and has an ocean of documentation behind it. I have a hard time figuring out what I need to start with or read into first. With that said, I strangely had a difficult time with some of the styling used in the dashboard. For example, I wanted to position the logo in the top-left of the dashboard. But doing so requires the title of the dashboard and the logo to be in the same container or div. Doing so resulted in the sizing of the logo being strange and the title of the dashboard being off-centered. After some trial and error, I realized I could use the flex layout.

Other challenges I encountered included the radio buttons used. Again, I wanted to position these in a particular way with a title and was able to after some trial and error through adjusting the padding and margin of them. However, I had a difficult time figuring out how to make the radio buttons actually respond with the callback used. I initially set up the callback and could click the radio buttons, but the dashboard itself would not update when I did, or would throw an error. A very similar issue occurred when I tried creating the geolocation map – I would just get an error each time I started the dashboard. After looking into this error further, I realized that the starter code provided the wrong Input and Output, which needed to be updated. Regarding the radio button, I realized I could simply pass the filter\_type as the determining factor. This was realized when I reviewed [this](https://dash.plotly.com/dash-core-components/radioitems) documentation from Dash Plotly’s website.

## Contact Ryan Hoskins