# CS 340 README

## About the Project/Project Title

This project provides a Python module that enables basic CRUD (Create, Read, Update, Delete) operations for interacting with a MongoDB database. It was designed to connect a web application’s database with a client-side interface by allowing data manipulation using Python and the PyMongo library. The module allows interaction with animal records, specifically designed for an Animal Shelter system.

## Motivation

The project was developed as part of Module Four milestone to implement Create, Read, Update, and Delete functionality. In the earlier milestones, I focused on implementing Create and Read operations, while Update and Delete were added later. The motivation behind this project is to simplify database interactions, support code reusability, and follow object-oriented programming best practices. The goal is to make it easier for developers to work with MongoDB through the use of a clean and reusable Python interface.

## Getting Started

To use this CRUD Python module, the MongoDB database must be set up and authenticated with the correct credentials. For the current project setup, I configured MongoDB to use a specific user (aacuser) with a password, connected to a MongoDB instance hosted on Apporto.

During the development of the CRUD functions, I used the Jupyter Notebook. The Create function allows users to add new documents (animal records) to the database, while the Read function retrieves and displays data based on search criteria. The Update function allows users to update documents to the database, while the delete function allows them to specifically delete certain documents.

Some challenges I encountered included:

1. Ensuring the Read function returns results in a readable format while excluding the MongoDB-generated \_id field from the output.
2. Testing the code in a regular Python script (.py) rather than in Jupyter. Since Jupyter handles certain execution steps differently, I had to be mindful of imports and how the AnimalShelter class was instantiated.

## Installation

For this assignment, I used a virtual lab that already included Jupyter Notebook and MongoDB. The main tools I used were MongoDB and the Jupyter Notebook. MongoDB is a database that has a way to store data, which was ideal for handling the animal shelter data in this project. It allows storing documents in a collection, making it easier to manage data. Jupyter was used to interact with the database. This library made it simple to connect to the MongoDB instance, perform CRUD operations, and retrieve data in a format that is easy to work with in Python. Since the virtual lab environment already had MongoDB and Jupyter set up, I was able to focus on the coding aspect without worrying about installation or configuration issues.

## Usage

The AnimalShelter class provides methods for interacting with the MongoDB database, allowing me to perform CRUD operations on an "animals" collection. Here’s how it works:

* Create: Inserts a new animal record into the database.
* Read: Retrieves records based on search criteria (for example, by animal name or type).
* Update: Updates an existing animal record based on specified criteria.
* Delete: Deletes an animal record from the database based on specified criteria.

### Code Example

*from pymongo import MongoClient*

*from bson.objectid import ObjectId*

*class AnimalShelter:*

*""" CRUD operations for Animal collection in MongoDB """*

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

*# Initializing the MongoClient. This helps to*

*# access the MongoDB databases and collections.*

*# This is hard-wired to use the aac database, the*

*# animals collection, and the aac user.*

*# Definitions of the connection string variables are*

*# unique to the individual Apporto environment.*

*#*

*# You must edit the connection variables below to reflect*

*# your own instance of MongoDB!*

*#*

*# Connection Variables*

*USER = 'aacuser'*

*PASS = 'GonSNHU1234' #Made sure this is correct based on the account that was created*

*HOST = 'nv-desktop-services.apporto.com'*

*PORT = 34053*

*DB = 'AAC'*

*COL = 'animals'*

*# Initialize Connection*

*self.client = MongoClient(f'mongodb://{USER}:{PASS}@{HOST}:{PORT}')*

*self.database = self.client[DB]*

*self.collection = self.database[COL]*

*def create(self, data):*

*""" Inserts a document into the MongoDB collection """*

*if data is not None:*

*result = self.database.animals.insert\_one(data)*

*return result.inserted\_id*

*else:*

*raise Exception("Nothing to save, because data parameter is empty")*

*def read(self, readData):*

*""" readData for documents from the MongoDB collection """*

*if readData:*

*data = self.database.animals.find(readData, {"\_id": False})*

*else:*

*data = self.database.animals.find({}, {"\_id": False})*

*return data*

*return list(data)*

*def update(self, query, update\_data, multiple=False):*

*""" Updates document(s) in the MongoDB collection """*

*if query and update\_data:*

*if multiple:*

*result = self.database.animals.update\_many(query, {"$set": update\_data})*

*else:*

*result = self.database.animals.update\_one(query, {"$set": update\_data})*

*return result.modified\_count*

*else:*

*raise Exception("Query and update data parameters cannot be empty")*

*def delete(self, query):*

*""" Deletes document(s) from the MongoDB collection """*

*if query:*

*result = self.database.animals.delete\_many(query)*

*return result.deleted\_count*

*else:*

*raise Exception("Query parameter cannot be empty")*

### Tests

### *from animal\_shelter import AnimalShelter*

### *# Instantiate an object from AnimalShelter*

### *test = AnimalShelter()*

### *# Test Create Method*

### *animal = {"name": "Kitty", "animal\_type": "Cat", "breed": "Persian", "color": "White", "age": 2}*

### *test.create(animal)*

### *results = test.read({"name": "Kitty"})*

### *for doc in results:*

### *print(doc)*

### *# Test Update Method - Change Kitty's color from White to Orange*

### *update\_query = {"name": "Kitty"}*

### *update\_data = {"color": "Orange"}*

### *test.update(update\_query, update\_data)*

### *# Read updated document*

### *results = test.read({"name": "Kitty"})*

### *for doc in results:*

### *print(doc)*

### *# Test Delete Method - Remove Kitty from the collection*

### *test.delete({"name": "Kitty"})*

### *# Verify deletion*

### *results = test.read({"name": "Kitty"})*

### *if not results:*

### *print("Document successfully deleted.")*

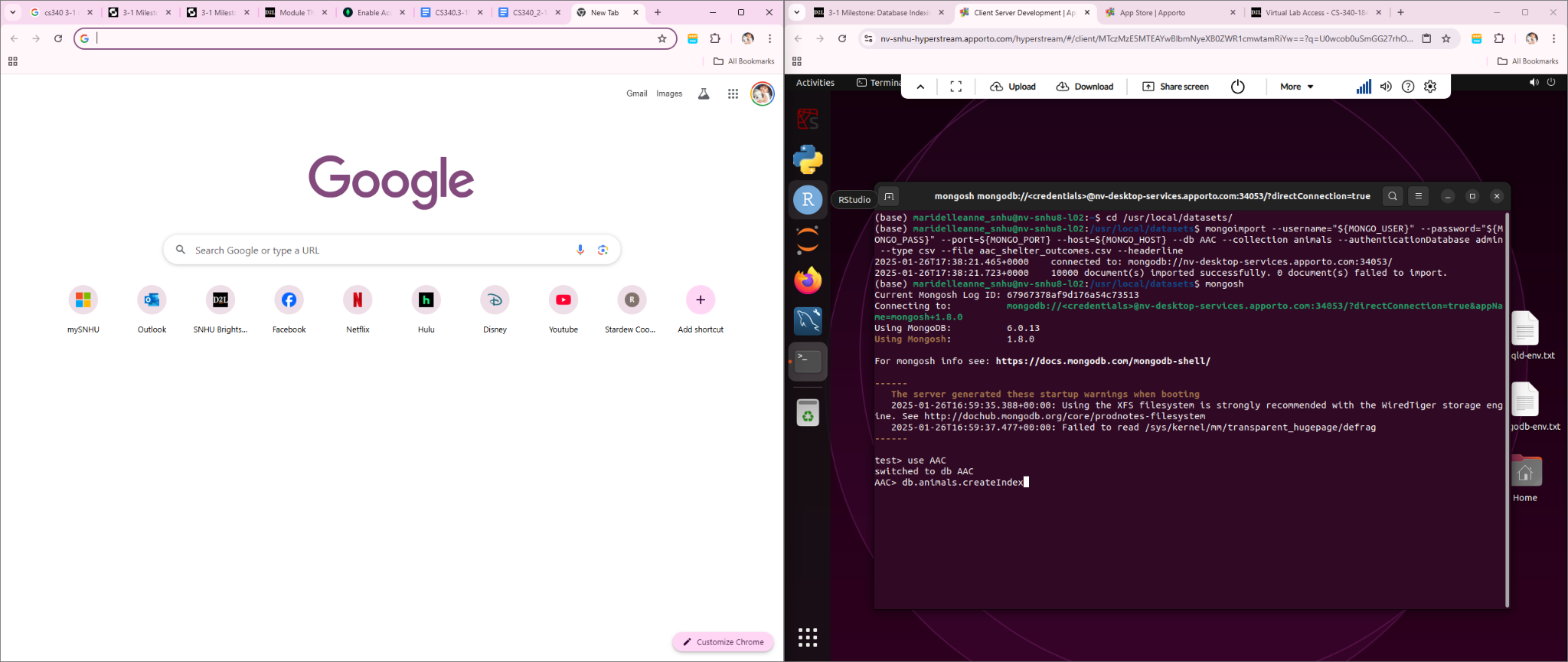
### *else:*

### *for doc in results:*

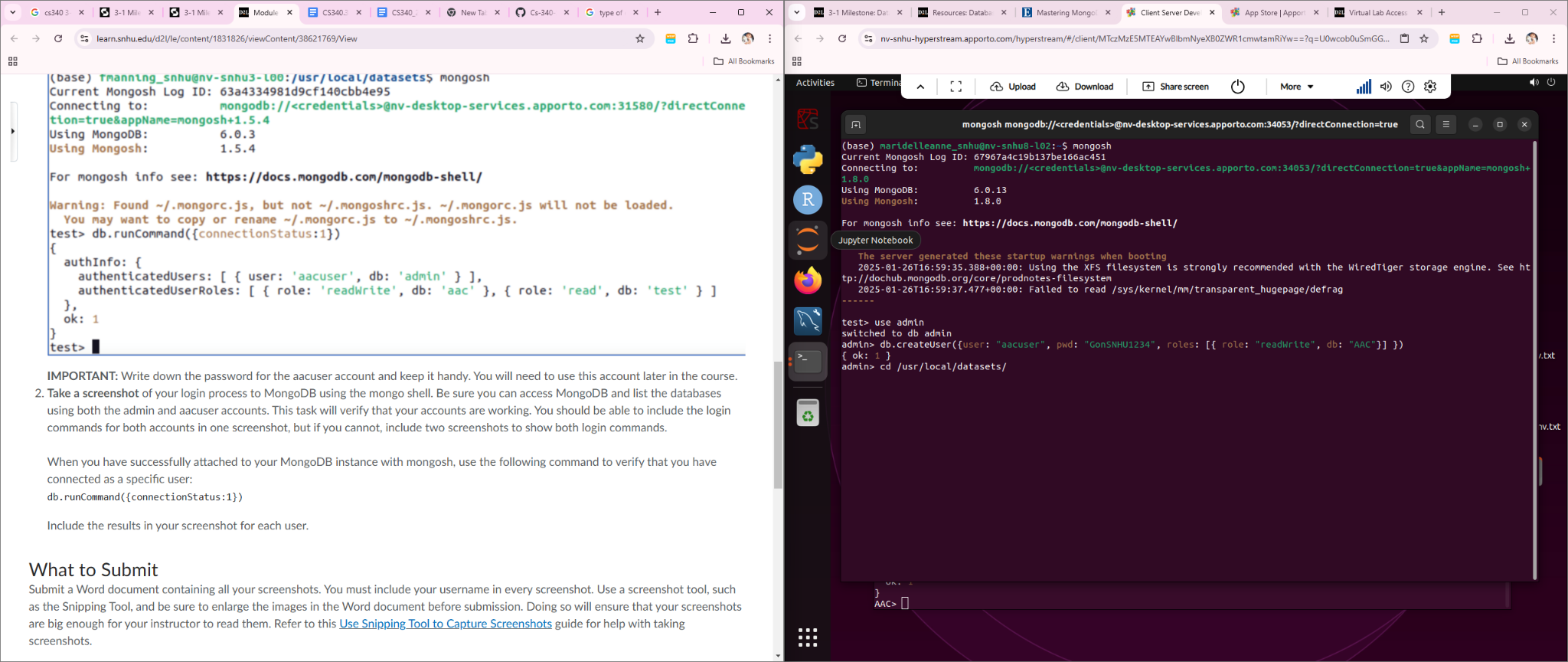
### *print(doc) # If something prints, the delete didn't work*

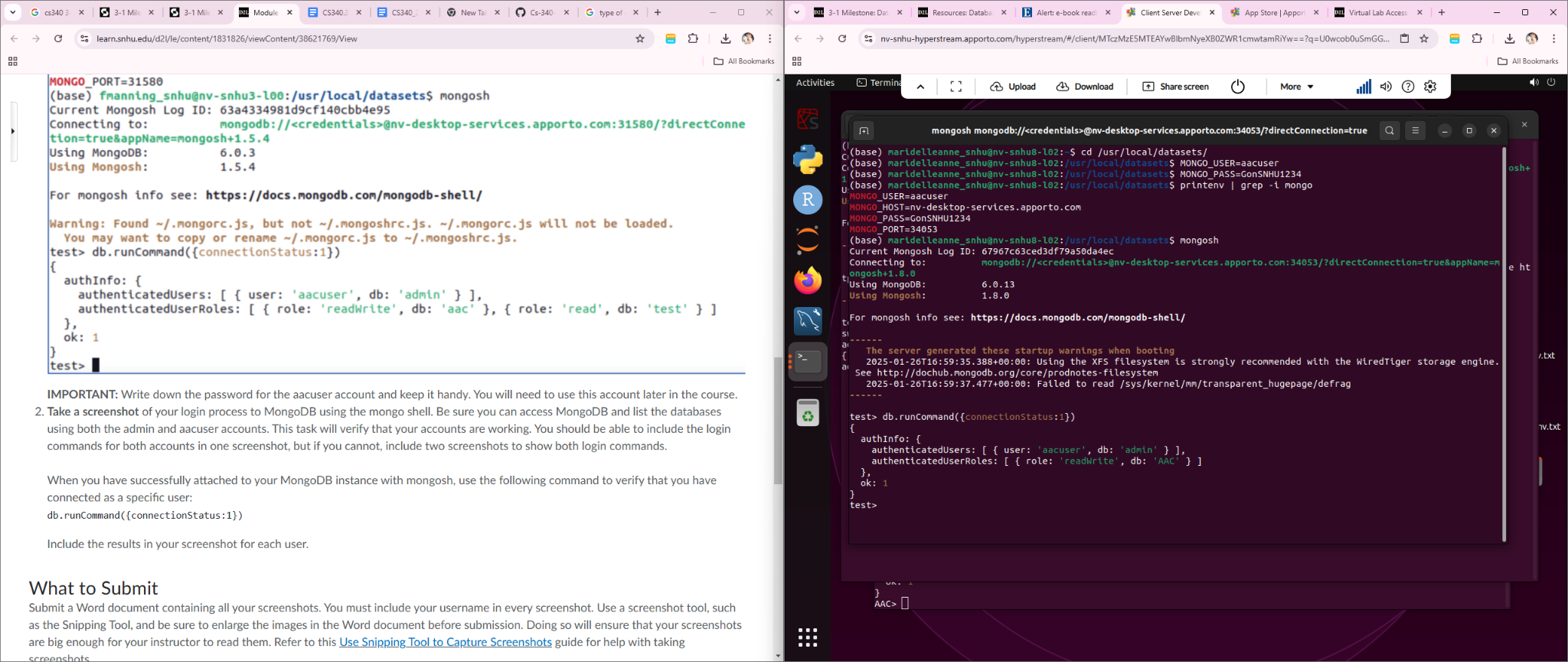
### Screenshots

*Upload the Austin Animal Center Outcomes data set into MongoDB by inserting a CSV file using the appropriate MongoDB import tool.*

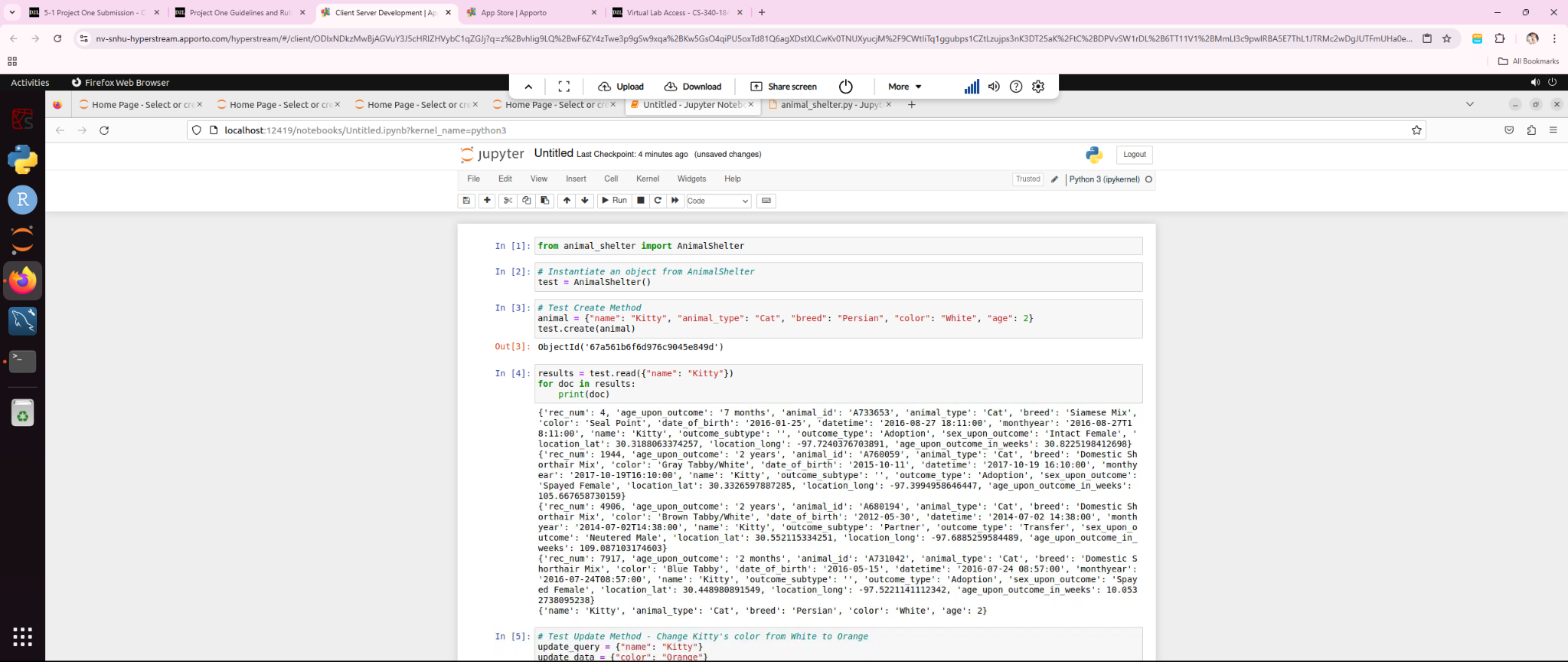
****

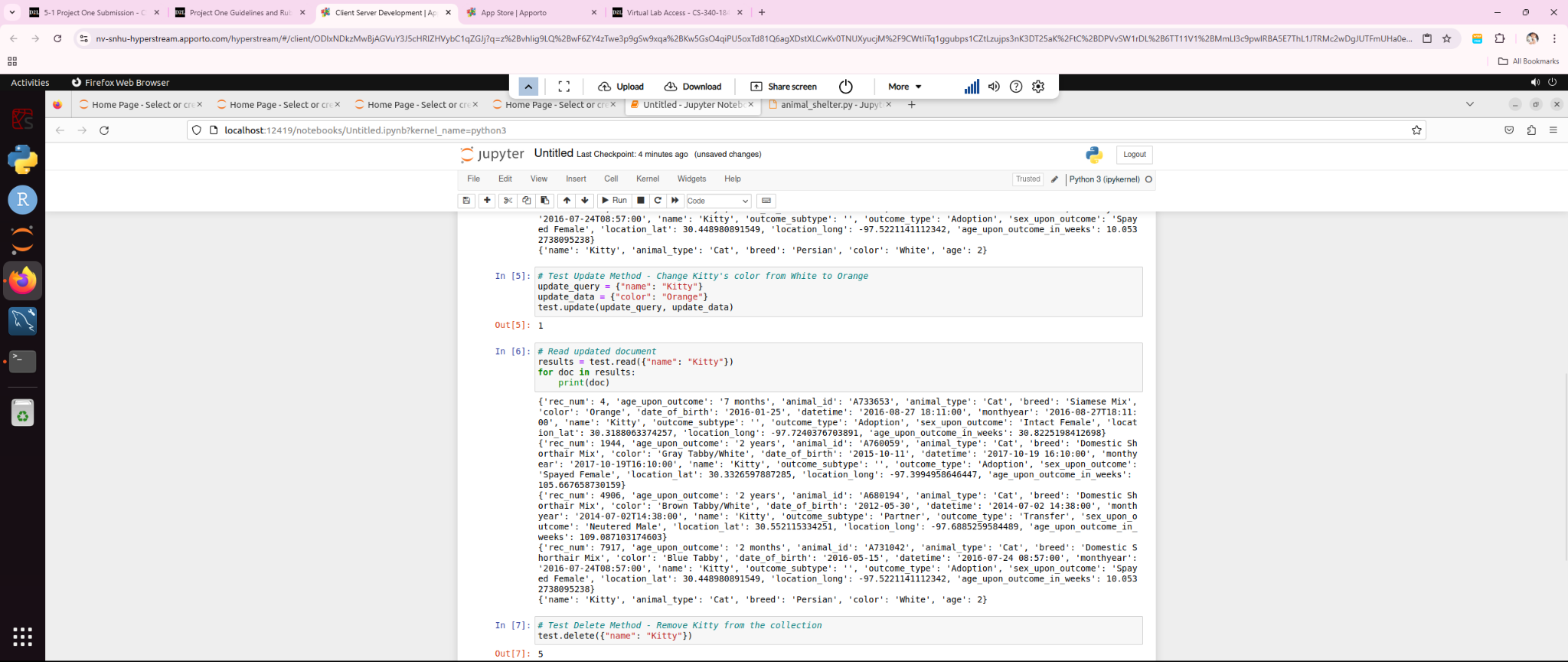
*Create a user account in the mongo shell to ensure user authentication to the database and collection you created.*

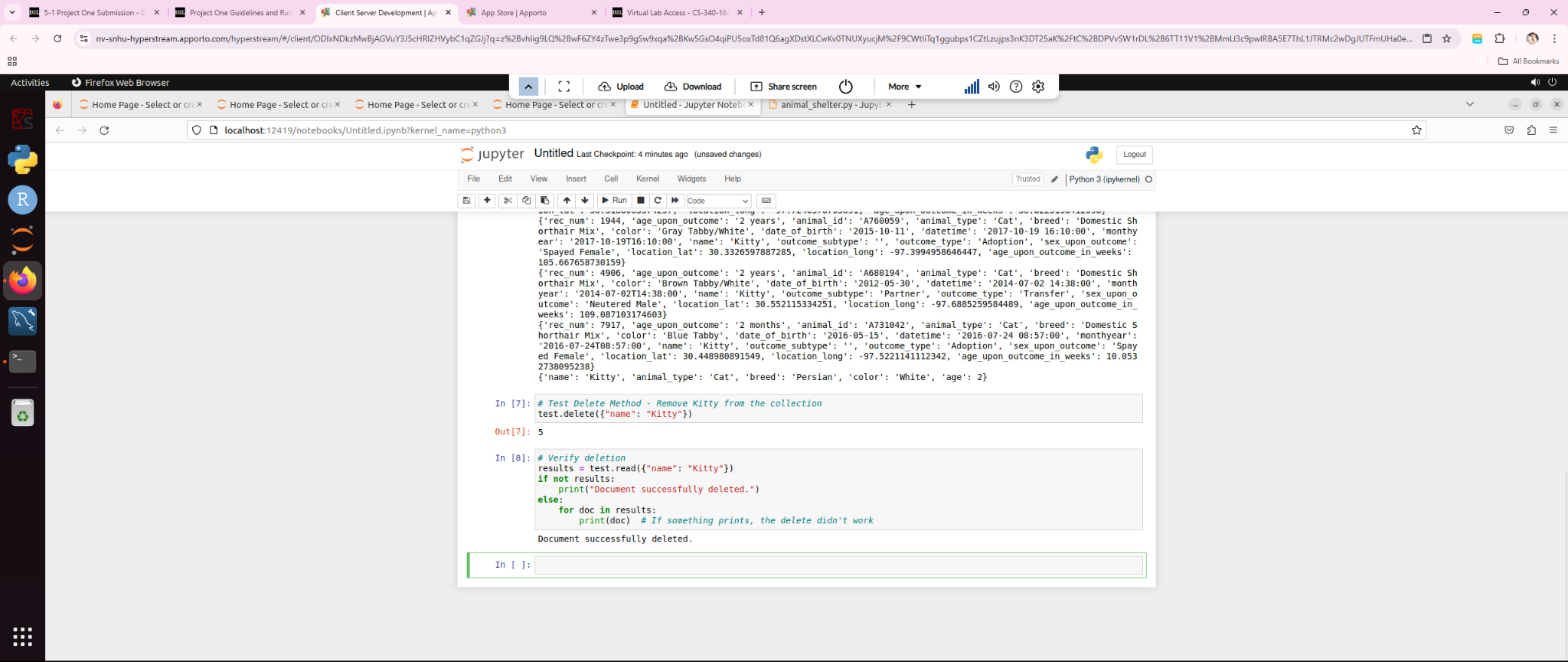
****



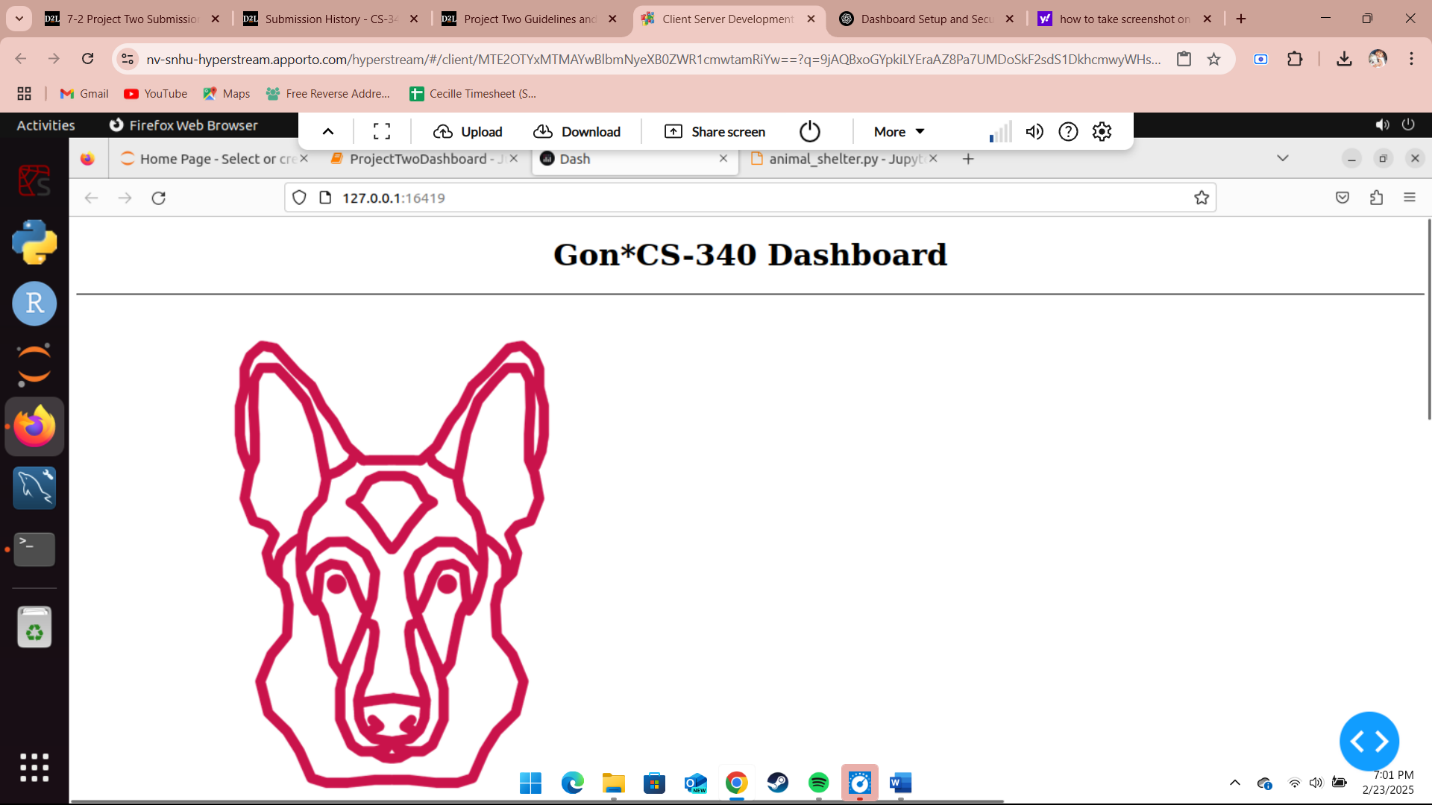
*Finally, you must test your Python module to make sure it works. To do this testing, create a Python script that imports your CRUD Python module to call and test all instances of CRUD functionality.*

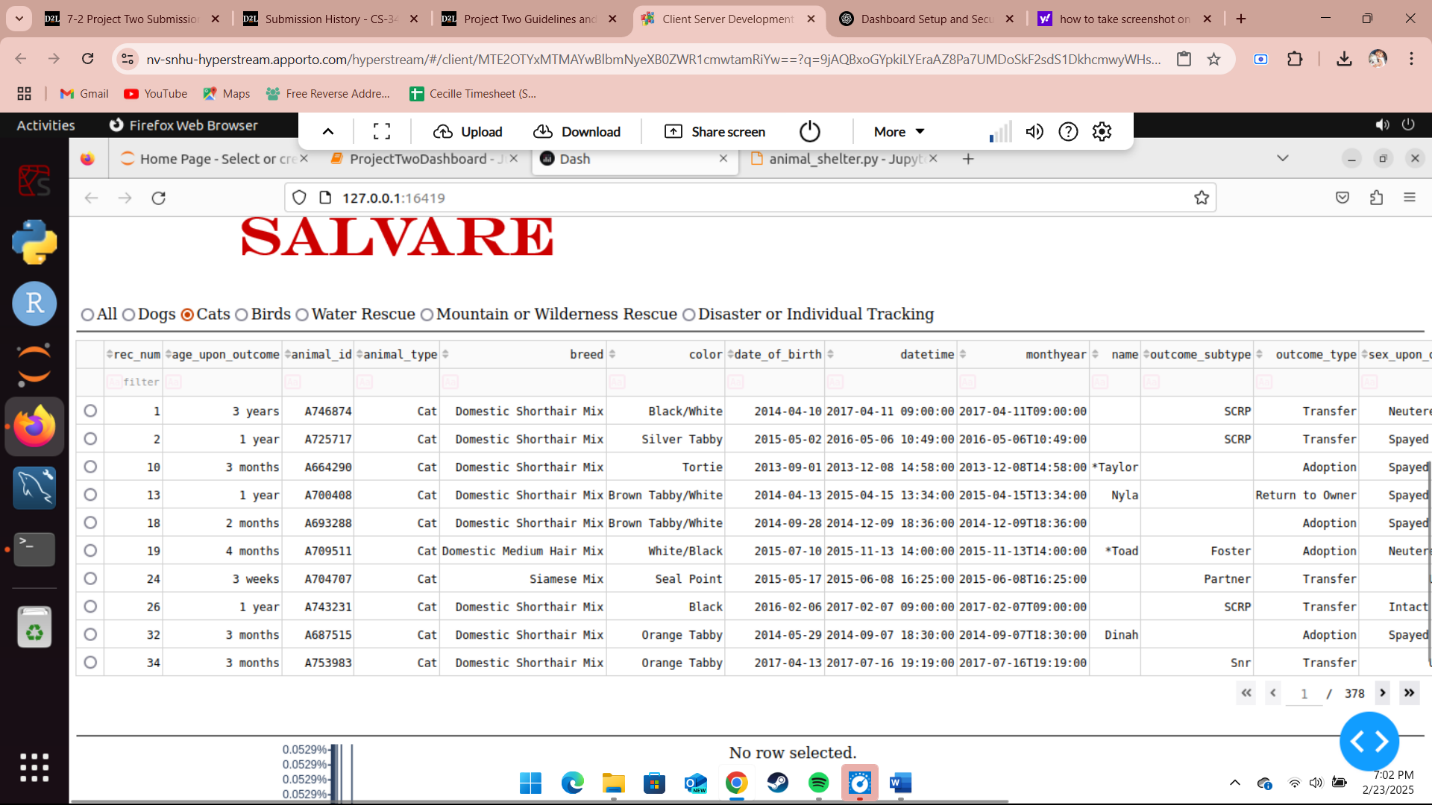
****

****

****

Dashboard





A screenshot of a computer

AI-generated content may be incorrect.

To complete this project, I took a series ofsteps to ensure the application functioned as expected. First, I set up a MongoDB database to store animal records, configuring a user account for proper authentication and authorization. I created a collection named "animals" where the records would be stored. I then developed Python functions for performing CRUD operations using the PyMongo library. I tested these functions in Jupyter Notebook to make sure they interacted with the database correctly.

Then, I used Dash to create a web application that provided a user-friendly interface for people to interact with the system. I added interactive components such as buttons and input fields, which were linked to the backend CRUD functions using Dash’s callback mechanism. This allowed real-time updates to the interface without the need for page reloads. Once everything was in place, I deployed the Dash application locally and tested it with various animal records.

Throughout the development process, I encountered several challenges. One challenge was ensuring that the data returned by the Read function was formatted correctly and excluded the MongoDB-generated \_id field. I resolved this by using PyMongo’s projection feature, which allowed me to specify which fields to include or exclude in the query results. Another challenge was integrating Dash with MongoDB operations. Dash’s interactive components needed to be connected to the backend functions in such a way that they updated the web interface dynamically. I overcame this by using Dash’s callback mechanism, which triggered Python functions when the user interacted with the interface, allowing seamless real-time updates.

## Contact

Maridelle Gonzales