Gregoria Ramirez  
CS 499 Capstone  
Module 5 – Enhancement Three: Databases  
July 27, 2025  
Dr. Troy Hawk

For my CS 499 Capstone, I selected the database integration portion of my Animal Shelter Dashboard project, originally developed in February 2025 for CS 340: Client-Server Development. In the original version, I used hardcoded MongoDB credentials and manually accessed data through the Mongo shell. While the application functioned, it lacked secure credential management, proper error handling, and modular database logic.

I chose this artifact because it clearly shows how I have improved in working with databases, especially in how I handle connections, queries, and security. Originally, I was using direct Mongo shell commands like this:

**# Original basic setup**

*from pymongo import MongoClient*

*from bson.objectid import ObjectId*

*class AnimalShelter:*

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

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

*# MongoDB Connection Variables*

*USER = 'aacuser'*

*PASS = 'SNHU1234'*

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

*PORT = 32440*

*DB = 'AAC'*

*COL = 'animals'*

**# Initialize MongoDB Connection**

*self.client = MongoClient('mongodb://%s:%s@%s:%d' % (USER, PASS, HOST, PORT))*

*self.database = self.client['%s' % (DB)]*

*self.collection = self.database['%s' % (COL)]*

While this worked for testing, it was not secure, and all database logic was mixed in with other parts of the code. For the enhancement, I replaced the hardcoded URI with an environment variable and refactored everything into a class called AnimalShelter to handle all the CRUD operations, including reading, filtering, and inserting records. This made the code more organized and reusable, and it kept the database logic separate from the rest of the application.

**# Enhanced version using secure connection and modular design**

*import os*

*from pymongo import MongoClient*

*from dotenv import load\_dotenv*

*load\_dotenv()*

*class AnimalShelter:*

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

*self.client = MongoClient(os.getenv('MONGO\_URI'))*

*self.database = self.client['AAC']*

In this enhanced version, I used python-dotenv to load the MongoDB URI from a .env file, which keeps the database credentials hidden and protected instead of hardcoding them into the script. This change makes the database connection more secure and easier to manage, especially if I need to deploy the app in different environments.

I also refactored the database connection into a class called AnimalShelter, which handles all CRUD operations. Putting the database logic into a class made the code more modular, reusable, and easier to maintain. It also keeps the main application cleaner by separating out the backend logic. To prevent the app from crashing when the database is unavailable, I added error handling to catch connection issues early:

*try:*

*self.client = MongoClient(os.getenv('MONGO\_URI'), serverSelectionTimeoutMS=5000)*

*self.client.server\_info() # Triggers exception if connection fails*

*except Exception as e:*

*print(f"Database connection error: {e}")*

Using python-dotenv to store the URI and creating a dedicated class for CRUD operations made the code cleaner, more secure, and ready for real-world use. I also implemented structured logging, improved error handling, and ensured the MongoDB client closes after each operation to prevent resource leaks.

**This artifact demonstrates my ability to:**

* Connect securely to a MongoDB database using Python.
* Manage sensitive credentials using environment variables.
* Modularize and reuse database access logic across the project.
* Implement logging and error handling for production readiness.
* Prevent connection leaks by closing MongoDB clients reliably.

**Key database improvements include:**

* Connecting to MongoDB using pymongo with structured exception handling.
* Loading all credentials from a .env file using python-dotenv.
* Dynamically building the MongoDB URI based on development or production context.
* Wrapping the entire database operation inside a reusable function or class method.
* Logging successful connections and safely handling errors.
* Removing the internal \_id field to prepare data for presentation in Dash.

**This enhancement supports:**

* **Outcome 4 (Computing Tools and Practices):** Applied tools like pymongo, dotenv, and pandas to transform and serve real-time data from a database into an interactive dashboard. This was achieved by using pymongo to securely query MongoDB, processing and cleaning the results with pandas, and integrating the data into the Dash dashboard for live filtering and visualization.
* **Outcome 5 (Databases):** Developed a security mindset by implementing secure credential storage, exception handling, and proper connection cleanup for MongoDB using Python. This was achieved by replacing insecure, hardcoded credentials with environment variables loaded through python-dotenv, adding structured error handling, and ensuring the database client closes after each operation.

In the future, I could strengthen the database layer by implementing role-based access control (RBAC) in MongoDB to limit user privileges, enabling SSL/TLS encryption for all data in transit, and encrypting sensitive fields at rest. I could integrate monitoring tools to track query performance, add automated database backups, , and also implement input validation at both the application and database levels to prevent injection attacks.