Gregoria Ramirez

CS 499 Capstone

Module 5 – Enhancement Three: Databases

July 27, 2025

Dr. Troy Hawk

For the databases category, I selected the backend code from my Animal Shelter Dashboard project. This project was first created in CS 340 using Jupyter Notebook and connected to MongoDB using shell commands and hardcoded credentials. For my CS 499 Capstone, I enhanced this part of the project to make the database interactions more secure, reusable, and professional.

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*

*client = MongoClient('mongodb://localhost:27017')*

*db = client['AAC']*

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. These changes align with Outcome 4 (implementing computing solutions with proper tools and practices) and Outcome 5 (developing a security mindset).

While enhancing this artifact, I learned how important it is to treat the database layer as its own part of the system , not just something you connect to quickly. It took some effort to figure out how to structure the class without repeating code and how to safely load the URI without breaking the app. But doing this helped me understand how to apply professional database practices, including connection safety and clean architecture.

This enhancement made my project more organized and secure, and it shows my growth in working with NoSQL databases like MongoDB in a real-world environment. I now feel more confident managing database logic in Python using best practices.