Nathan Johnson

SNHU

CS-499-13459-M01

5-2 Milestone Four: Enhancement Three: Databases

04/6/2025

Enhancement Three: Databases

Artifact Description

I chose a database interaction module from CS 340 called animal_shelter.py. This program, built in Python, connects to a MongoDB database and manages animal shelter data. It provides basic CRUD operations (Create, Read, Update, Delete) for adding new animal records, updating existing information, deleting records, and retrieving specific data. The original implementation used hardcoded database credentials and had minimal error handling. The program successfully connected to the database and performed basic functions but lacked security features and performance optimizations that would be needed in a real production environment.

Justify the Inclusion of the Artifact

The MongoDB connection code worked, but security was a major issue. The hardcoded credentials in the source code meant anyone with access to the file could see the database username and password. Error handling was minimal, with operations sometimes failing silently or providing unhelpful error messages. The database queries weren't optimized, which would cause problems as data volume grew. There was also no testing framework to verify the code worked properly. That made it perfect for improvement. I saw a chance to take something functional but insecure and transform it into something that could handle real-world scenarios with proper security and performance.

Improvements and Enhancements

I improved the system by replacing hardcoded credentials with environment variables. This makes the application more secure by keeping sensitive information out of the source code. Now credentials can be managed separately for different environments (development, testing, production), following security best practices for database applications. I added comprehensive error handling with try-except blocks around database operations. The enhanced code now catches specific MongoDB exceptions and provides clear error messages. I also added logging functionality that records errors, warnings and successful operations. This makes the system easier to debug and monitor, especially when things go wrong in production. Through adding unit tests for all CRUD operations, I made the system more reliable. The tests use a separate test database to avoid affecting real data and include setup and teardown methods. This ensures each function works correctly and maintains data integrity. Testing also makes it safer to make future changes since we can quickly verify nothing broke. I put in database indexing too. By adding indexes to frequently queried fields using MongoDB's createIndex() method, searches are much faster, especially for operations filtering on common fields like breed and location. The query optimization ensures good performance even with larger datasets, providing a more responsive experience for users.

Course Outcomes

I fixed this artifact and did everything I planned to do. I wanted to make the system more secure by implementing proper credential management. I wanted to add error handling to make it more reliable. I wanted to add tests to verify functionality. I wanted to optimize database queries for better performance. I did all these things just like I intended to do.

The improvements align with Course Outcome 5 by developing a security mindset that anticipates potential vulnerabilities. Removing hardcoded credentials protects sensitive data from exposure. The proper error handling prevents information leakage that could happen with generic error messages. The logging system creates an audit trail that can help detect suspicious activity. The enhanced code with its optimized queries demonstrates alignment with Course Outcome 3.

By adding database indexes, I showed how to design solutions that manage trade-offs between query performance and storage requirements. The improvement in search speed shows algorithmic thinking applied to database operations. The testing framework demonstrates alignment with Course Outcome 4 by showing well-founded database techniques that deliver value in real-world scenarios. Unit tests that verify CRUD functionality ensure data integrity and system reliability, important requirements for any production database.

Reflection on Process

Database security matters in software development. When credentials are stored as environment variables, they're protected from accidental exposure in source control. Getting this to work properly required understanding how environment variables are managed in different systems. Looking back, I should have considered even more security measures like connection pooling, though the current implementation provides a good balance of security for most use cases. Error handling proved more complex than expected. MongoDB can throw different types of exceptions, and properly catching each type required studying both Python's exception system and MongoDB's error documentation. The most challenging part was deciding what constitutes a user error versus a system error and providing appropriate responses for each. Adding database indexes was straightforward but required thinking about query patterns. Not every field needs an index, and too many indexes can slow down write operations. Finding the right balance meant analyzing the most common queries and focusing on those fields. The performance gains were significant, operations that used to scan the entire collection now use indexes for direct lookups. Time complexity and space requirements don't always align. Indexes speed up queries but take extra storage space. Testing showed the trade-off was worth it. Functions produced the same results, just with better efficiency. This project taught me how important query optimization is for database systems, and how important a skill it is for any software engineer working with data-intensive applications. The enhanced database module now has the security, reliability, and performance features needed for a real production environment.