CS 499 Computer Science Capstone

3-2 Milestone Two: Enhancement One: Software Design and Engineering

Professor Dr. Pravin Bhandari

Mubeen Ahmed Khan

mubeenahmed.khan@snhu.edu

Southern New Hampshire University

21st September, 2025

Enhancement One – Software Design & Engineering

Artifact: animal_shelter.py (MongoDB CRUD module)

Overview of the Artifact

The artifact is a Python CRUD module, animal_shelter.py, used by my CS-340 Dash/Jupyter dashboard to interact with a MongoDB dataset of shelter animals. The enhanced version preserves the original public API (class and method names/signatures) while adding professional engineering features: docstrings and type hints, centralized logging with user-friendly messages, optional environment-driven configuration, fast connection checks (timeouts + ping), and conservative input validation (including a safe allow-list for update operators and support for `read({})` to return all documents).

For context, the original baseline provided basic CRUD with minimal error handling and without structured logging or configuration controls.

Why I Included This Artifact

I selected this artifact because it shows practical software design applied to a real data workflow: an analytics UI backed by MongoDB through a reusable Python module. Key improvements demonstrate:

- **Quality & clarity:** Comprehensive docstrings and type hints that clarify intent and improve maintainability.
- **Observability:** Central logging `(logging.basicConfig(...))` plus friendly `print(...)` messages for notebook users.
- **Portability:** Optional env-driven configuration `(MONGO_URI, MONGO_DB, MONGO_COLL)` while retaining the original username/password path.

- **Reliability & safety:** Fast connection validation; `_validate_filter(..., allow_empty)` so `read({})` works with the dashboard; strict, operator-safe filters for update/delete; update allow-list and $-key guards.

(An earlier enhanced draft helped validate these changes during integration.)

Course Outcomes Addressed

- **Software engineering/design/database:** I applied configuration management, structured logging, input validation, and connection health checks to deliver a robust, maintainable CRUD layer—without breaking existing consumers.
- **Professional communication:** Clear docstrings and consistent logs communicate behavior to collaborators and provide helpful user feedback in notebooks.
- **Problem-solving with trade-offs:** Allowing empty filters only for reads (to support db.read({})) while keeping writes strict reflects deliberate design trade-offs between usability and safety.

Reflection on the Enhancement Process

What I learned. I reinforced PyMongo best practices (e.g., avoid truthiness checks on Collection; use timeouts and ping) and designed validation that fits the product context—supporting all-records reads for dashboards without weakening write safety.

Challenges and Resolutions.

- **Notebook compatibility:** Early strict validation rejected {}; adding `_validate_filter(..., allow_empty=True)` resolved this for reads while keeping writes protected.
- **Library nuance:** Replacing `if not self.collection:` with `if self.collection is None:` avoided PyMongo's `NotImplementedError` for truthiness checks.

- **Environment parity:** Case sensitivity (e.g., aac vs AAC) can cause write errors on some systems; aligning DB names and supporting env overrides eliminated these issues.

How the Artifact Was Improved (Concrete Changes)

- Added docstrings/type hints and central logging; preserved friendly `print(...)` messages for immediate UI feedback.
- Enabled env-driven configuration (URI/DB/COL) and fast-fail connection checks.
- Implemented filter validation with `allow_empty` support for reads and a safe update allow-list with $-key guards.
- Preserved the original class/method structure for drop-in compatibility with dashboards and tests.

Possible Indicators of Success (Rubric Alignment)

- **Innovative techniques:** env-configurable connections, health checks, structured logging, and safe validators.
- **Programming solutions:** balanced `read({})` support vs. strict write validation and operator allow-listing.
- **Security-aware design:** blocks top-level $ operators, prevents $-prefixed keys, restricts updates to a safe set.
- **Clear articulation via a working product:** enhanced module runs with the CS-340 dashboard and communicates status/errors clearly.