Artifact Enhancement Three Narrative

Anthony Baratti

Southern New Hampshire University

CS-499 Computer Science Capstone

Professor J. Lusby

August 3, 2025

Artifact Enhancement Three Narrative

1. Briefly describe the artifact. What is it? When was it created?

The artifact for enhancement three is the Grazioso Salvare Animal Shelter Dashboard. It takes a .csv file loaded with 10,000 animals (different types, breeds, locations with lat and long, etc) and populates a database in MongoDB, then filters the results using CRUD operations to display them in meaningful ways to the user (such as a data table, a pie chart, and an interactive geo-location map). It was created in CS-340 Client/Server Development in December of 2024.

2. Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in software development? How was the artifact improved?

The artifact was included to perform changes to the database structure to satisfy enhancement three's category (databases). The ability to convert a database from the JSON structure that MongoDB uses to an SQL model that SQLite 3 uses was featured in the enhancement. The artifact was improved by removing the need for online resources, localizing the database, and the execution environment (which is an important process for programming mobile applications). Converting the JSON formatting to the SQL format showed that complex queries could be performed (filtering multiple keys to return multiple filtering options via AND and IN keywords) on the SQLite database. All of the CRUD operations were created and tested, but the dashboard only uses the read functionality to pull the data from the database and display it using Dash and Plotly. Furthermore, the program was turned into a standalone .exe file using PyInstaller, which means that it can be downloaded to a Windows system and the user can simply click the .exe, which will open a web browser, direct to the appropriate offline address, and run the program without any command line prompts. The database was preloaded before the program was packaged, so there is no need to run the script that populates either. Just “click and go”.

3. Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

(1). Refactoring code to SQLite allows for the database to be run locally and stay lightweight, which can help reduce overhead and requires minimal setup for functionality. This helps to foster a collaborative environment where diverse audiences can easily operate and manage the program.

(3). By redesigning the CRUD operations for SQLite, we can manage small and agile local applications via a web browser with an HTML graphical user interface. While this would not be ideal for multiple locations, it would work perfectly for strictly single on-site locations. Evaluating requirements is a crucial step towards designing and implementing appropriate computing solutions while managing their trade-offs.

(4). By combining Dash libraries and extensions with Python, SQLite, and HTML coding, we have utilized tools to increase efficiency to achieve the goals of the system. This delivers appropriate and valuable solutions to tackle industry-specific problems.

(5). By scripting the SQL clause format into CRUD operations appropriately (such as “?” as value placeholders), we can mitigate any potential for SQL injection. While the data being entered is predefined (i.e., radio buttons prevent users from directly inputting text), the appropriate security measure of using placeholders to build the initial table showcases security awareness. Furthermore, keeping the database localized (saved as a .db file), we can restrict access from external sources. This achieves a security mindset for SQLite3 conversion. Also, a test script has been provided to ensure the CRUD operations work as intended, testing the application for complete functionality of the read function on the dashboard. This helps reveal and fix any bugs using the errors and callbacks developer tab (in the web browser), as well as monitoring the GET/POST procedures revealed in the command window during operation.

4. Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?

One interesting thing that I learned while developing the SQL structure for database queries was the separation of concerns. The example I'll use here is the CRUD.py. Using a variable for the base query and a set of filters passed to it rather than making the call directly from the ShelterDashboard.py allowed for less and more readable code. The base query was the "SELECT * FROM animals" += " WHERE {filter}" which was passed from the radio button filters. Then, instead of using an index for the keys, I was able to return the dictionary with the column names and used the names ([row][columnName]) instead of the index, which is helpful if the returned key value pair did not match the index that was used. This would help prevent any out-of-bounds checking as well. One challenge that I did not anticipate was scripting the filter queries in the app.callbacks. When the radio button is selected, the filter needs to pass the pre-defined query back to the read function. The hard part was that I had a lot of typos and syntax errors. For example: "(animal_type = 'Dog' AND " "breed IN ('Labrador Retriever Mix', 'Chesapeake Bay Retriever', …….") I found that I was missing commas or quotation marks in random locations, which created errors during testing that were difficult to detect. I had to meticulously, character by character, go through each query to ensure perfect accuracy, even down to spaces such as " OR " with a space on either side of it to separate the query. One misplaced space, quotation, comma, parenthesis, or bracket would make the entire query fail. Attention to detail, patience, and troubleshooting were vital in the completion of this enhancement.