Final Combined Narrative

Curtis R Thomas

Southern New Hampshire University

# 

# Combined Narrative on the Artifact

What follows is a summation of the three narratives submitted in the course, re-worked and reflecting the finished project.

# Describing the Artifact

This artifact is a backend API that includes logic for building and managing tables in a postgres database, as well as logic for supplying URLs in order to make the API consumable for a website or web service. It makes use of various useful, in-demand technologies including FastAPI, sqlalchemy, Postgresql, OAuth2 and JWT for security, and Pydantic for schema control. This project also includes unit tests to test for functionality while development is ongoing. This project was developed over the course of eight weeks with an estimated total of 40-60 cumulative hours of effort. This artifact is being used to represent my growth in Software Engineering, Algorithms and Data Structures, as well as Database usage.

## Justification for Inclusion On the Basis of Software Engineering

The model-schema-crud components of this artifact demonstrate my growth as a programmer and deserve to be included because they demonstrate a more modular, easy-to-change design compared to the PyAlaMongo artifact. This is very similar to the design pattern I learned in my job, and I’ve come to greatly appreciate it for its modularity, being both easy to re-use and easy to change. While not as simplistic as the PyAlaMongo artifact, it’s also more robust against bad data (thanks to Pydantic schemas) and better able to adapt to different unforeseen needs in business logic. Additionally, the implementation of unit testing demonstrates growth insofar as I struggled with unit testing earlier in my school career. This artifact also demonstrates growth through my use of clear, descriptive naming conventions.

**What Existed Before, and How It Was Enhanced.** What was present before was a backend technology artifact called PyAlaMongo that used Pymongo for database management (crud calls, establishing documents), used native python as middleware, and used Dash API to receive data from the user. Documentation was relatively sparse, and there was no validation of incoming data or any security beyond a database login. This artifact has been enhanced through the implementation of sqlalchemy for database management, allowing flexibility between multiple database technologies if needed, as well as providing more robust controls in interfacing with the database; the implementation of models allows for more advanced use of data structures being returned from the database, such as accessing column values as object attributes; the implementation of schemas allows for some enforcement of data validation; the implementation of fastAPI over Dash divorces the project from the single front-end technology of Dash and makes it consumable by a variety of front ends and even applications while also enabling the use of more robust security such as OAuth2 tokens and permission checking at time of API consumption. .

**Course Objectives.** Through the use of ReadMes, supplemental text files, commented code, and plain, descriptive naming conventions, I feel that I am employing strategies that support diverse audiences in making decisions about software and business needs. In the use of comments and ReadMes, I am demonstrating audience-targeted, technically sound, and professional communication. Through the use of well-founded and innovative techniques, such as performing data validation in the schema layer or providing programmatic database management through sqlalchemy, I am developing computer solutions that deliver value and accomplish industry specific goals. Through the use of default-deny permissions checking on endpoints, storage of hashed passwords instead of plaintext passwords, abstraction of important data (such as secret keys) into environment variables, and the use of OAuth2 logins, I am demonstrating a security mindset that aims to anticipate security threats and malicious actors. In accordance with my own objectives for this segment of the project, I have made the code more consistent with PEP-8 guidelines, made the code more robust to varying business needs by making it easier to change and abstracting out logic where possible, and made the code far more secure than it was in the PyMongo project.

***Process reflection.*** The development process of this artifact was much more difficult than I had anticipated. I was under the impression that I would be able to establish much of the boilerplate with the aid of a tool provided by fastAPI, but was not able to get it running correctly. I had no choice but to deal with this problem, because failing to do so would prevent me from following the roadmap I’d laid out for this course. Manually referencing materials to build boilerplate was a lengthy process and was perhaps the biggest challenge that I faced while building this artifact. However, manually building the boilerplate to allow this stack’s technologies to interact has also brought me a better understanding of the technology, and I feel more equipped than ever to continue my work with it.

## Justification For Inclusion on the Basis of Data Structures and Algorithms

The inclusion of this artifact shows my skills in data structures in several respects. The first exhibition of an advanced understanding of data structures is the use of string enumerations to be passed to the same enumeration type in the permissions schema class; the use of enumerations means that only a few select values can possibly be passed to the schema successfully. Next, the use of the JSON type to store the permissions affiliated with a user demonstrates thoughtful use of data structures instead of simply adding more and more string columns to track permissions. Finally, the use of the schemas to perform type-hinting, validation, and enforce enumerated values being passed from the endpoint to the database shows greatly advanced and improved handling of data structures over the previous artifact. In the pytests, the author demonstrates knowledge of hashing by using python dictionaries to rapidly compare values between expected and returned data. Some examples of algorithm use in this project include a method for returning events from the database sorted by a sort\_order attribute, as well as the use of logic for generating OAuth2 tokens and encrypting and decrypting passwords.

**What Existed Before and How was it Enhanced?** What was present before was a backend technology artifact called PyAlaMongo that used Pymongo for database management (crud calls, establishing documents), used native python as middleware, and used Dash API to receive data from the user. This artifact did not make use of advanced data structures or algorithms to achieve its business logic or to establish security. This artifact was enhanced through the implementation of more advanced data structures, the use of dictionary lookups where appropriate to optimize running time, and the use of algorithms to secure password data and generate secure web tokens.

**Meeting the course objectives.** The author planned to address data structures and algorithms through the use of filtering and sorting db query results, enforcing schemas on CRUD, and appropriate implementation of data structures in the model layer. All of these goals were achieved in this artifact. This segment of the project also demonstrates the use of strategies to enable diverse audiences to engage in collaborative decision-making by making the code easy-to-change and abstracting algorithmic logic into different functions so that it can be used or not used where desired, as well as by using appropriate and useful data structures that are also accessible to understand, explain, or interface with (such as JSON inputs and outputs). This artifact also demonstrates design of computer solutions that use appropriately balanced algorithmic solutions and computer science best practices to solve business needs where appropriate. This project demonstrates a security-forward mindset through the use of algorithms to generate secure web tokens and the use of hashing to keep password data safe while also retaining its utility.

***Lessons learned.*** Much of what the author has learned has to do with the tools used in this tech stack. For example, Alembic will fail to detect certain changes to columns during a migration, causing a pernicious type error. Additionally, the author gained in-depth experience in implementing endpoint security and user authentication through the use of algorithms, web tokens, secure password-checking and user headers. The author also learned the value of using enumeration types to enforce input checking on strings passed to the API.

## Justification for Inclusion on the Basis of Database Competencies

The inclusion of this artifact demonstrates the development of applied skills in working with databases. More than simply making a create, read, update, delete call, this API’s models have relationships between tables via an association table (the UserSubscriptions class in the User model), and makes advanced CRUD calls such as a query that also sorts data while reading. This artifact also demonstrates the use of the sqlalchemy library to perform programmatic database management that would be relatively easy to change between database technologies, the alembic library to perform database migrations, and the use of schemas and models to gain added utility from database objects (such as being able to access column data as attributes when an object is returned as a model from the database) as well as using schemas to perform data validation to enforce some data safety.

**What Existed Before and How Was It Enhanced?** What was present before was a backend technology artifact called PyAlaMongo that used Pymongo for database management (crud calls, establishing documents), used native python as middleware, and used Dash API to receive data from the user. This artifact was enhanced through the implementation of sqlalchemy for the database management technology, allowing for greater flexibility if a different database technology is desired, as well as more robust options for interfacing with the database programmatically. Data validation is now enforced through schemas where there was no data validation before, and users simply had to be trusted to not pass bad data.

**Meeting objectives.** With respect to my own goals for this segment of the project, one of the objectives established in the module one document was to establish a postgres backend, which has been accomplished. Another item I suggested was using database triggers. As this project has come into focus, it has become clear that there was no appropriate use-case for using triggers (which could have been established on the models using sqlalchemy); instead, I demonstrated advanced db concepts through the use of association tables (app.models.user: UserSubscriptions) to link event rows and user rows where associated by the user id. With respect to course outcomes, I developed effective and professional communication with specific audiences and contexts through the use of comments and supplemental documentation on the database layer, as well as the use of consistent, understandable, and plain naming conventions for columns to ensure broad accessibility. I have demonstrated the use of well-founded and innovative technologies, skills, and tools through the use of sqlalchemy for programmatic database management, as well as the use of pydantic schemas for data validation and models in order to facilitate reliable and useful communication of data to and from the database. I have demonstrated a security-forward mindset through storing passwords only in a hashed form and using default-deny permission checking where appropriate on endpoints that access the database.

***Reflecting on the development process.*** The development process for this segment was less difficult than other segments, and has been much more about refining the existing project to better reflect my skills and academic goals. My plans with respect to demonstrating database concepts and skills did have to change somewhat as the design of the program became better defined. One place that has presented a consistent challenge is writing better documentation; one solution implemented is the use of a readme and supplemental documentation for different layers of the program to help incoming programmers get a better high-level understanding of the design.