Chetachi Ezikeuzor  
Jun 1, 2025

Milestone Two: Narrative for Enhanced Artifact

The artifact I selected for my ePortfolio is *Travlr*, a full-stack travel planning application I built in early 2025 using Node.js, Express, MongoDB, and Angular. It allows users to register, log in, and manage their personal travel itineraries through a responsive interface. While the original version had a functional foundation, it lacked the architectural depth and maintainability expected in production-grade systems. I chose *Travlr* because it reflects the full range of technical skills I’ve gained during SNHU’s Computer Science program—especially in software design, architecture, security, and application layering.

Initially, the project’s structure was functional but rigid. Middleware was tightly coupled with routing logic, controller responsibilities were blurred, error handling was inconsistent, and documentation was minimal to nonexistent. As I reviewed my code through the lens of real-world engineering principles—and the feedback I received during Milestone One—I realized the app needed substantial internal restructuring. This milestone gave me the opportunity to take a working but raw prototype and turn it into something more robust, modular, and aligned with modern development standards.

Key Enhancements Implemented  
*Modularization and Separation of Concerns*

The most pressing structural issue was the lack of separation between logic layers. I addressed this by fully adopting an MVC architecture. Routes now live in dedicated files (e.g., trips.js, authentication.js), and controller logic is offloaded into a new controllers/ directory. I also moved middleware functions—including JWT authentication and input validation—into a separate middleware/ folder. This modular structure significantly improves readability and makes the app easier to maintain, test, and expand.

Security Enhancements

Security was a major focus. I replaced the original manual crypto-based password hashing with bcrypt, which securely handles both salting and hashing out of the box. I also switched the authentication flow to stateless JWTs, simplifying server-side logic. Every API endpoint now includes input validation and sanitization to guard against injection attacks and malformed data. For consistency and clarity, I centralized all HTTP status codes and error messages in a constants.js file.

Logging and Error Handling

The old project relied on console.log() for runtime output, which wasn’t sufficient. I integrated Winston, a structured logging library that outputs timestamped, leveled logs—ideal for debugging and production monitoring. I also standardized error handling across the app. Instead of scattered try-catch blocks, I now use centralized error handlers that log issues cleanly and return consistent response formats. This gives me clearer insight into what’s happening when things go wrong.

Data Normalization

In the original trips.ts, trip lengths were stored as strings like "4 nights / 5 days", which made filtering and sorting harder. I normalized this data by storing trip length as an integer number of days. That small change made the frontend filters easier to implement and leaves the door open for future enhancements like range queries or comparisons.

Comprehensive Documentation

I documented the application thoroughly using JSDoc. All controllers, middleware, and model methods now include clear descriptions of their purpose, parameters, behavior, and error-handling expectations. I also added inline comments where logic was complex. Not only did this make the code easier to navigate, but it also helped me debug and think through my own logic more clearly.

Alignment With Course Outcomes  
This enhancement aligns directly with the learning outcomes outlined in Module One:

* Innovative Software Engineering Practices: I applied modular design patterns, JWT-based authentication, centralized config management, and structured logging—hallmarks of modern engineering workflows.
* Design Flaw Resolution: I refactored monolithic files, decoupled layers of logic, and increased clarity and scalability across the codebase.
* Professional Communication & Documentation: Through meaningful documentation and error messaging, I improved the project’s long-term usability and handoff-readiness.

Reflection and Challenges

This process made me rethink what makes an app "good." It’s not just about whether it runs or looks polished—it’s about whether it’s secure, maintainable, and adaptable under pressure. Refactoring while keeping the core logic intact required planning and careful testing. Switching to JWTs forced me to rethink the login flow from start to finish. Adding Winston also introduced new questions—like what should be logged, at what level, and in which environments.

One of my biggest takeaways is that documentation and structure aren’t optional—they’re essential. These improvements changed how confident I feel about handing off my code or deploying it live. I also got hands-on experience balancing readability and function, which is a subtle but essential software engineering skill.

Final Thoughts

The updated version of *Travlr* represents a complete transformation. It’s no longer just a working prototype—it’s a secure, and properly architected application. This project reflects my growth as a developer and my ability to apply real-world engineering practices thoughtfully. It’s a strong, central piece in my ePortfolio because it doesn’t just show where I started—it shows where I’m headed next.