Chetachi Ezikeuzor  
April 6, 2025

Milestone One Addendum: Outcome Alignment and Existing Functionality

In my original video walkthrough for Milestone One, I focused on showcasing the structure of the Travlr application and outlining a thorough enhancement plan. However, I’d like to provide a more intentional breakdown of the existing functionality within each category—software engineering and design, algorithms and data structures, and databases—before enhancements were made. I’ll also clearly explain how my planned improvements align with the CS 499 course outcomes.

Software Engineering and Design – Existing Functionality & Outcome Alignment

At the time of the initial review, the backend structure of *Travlr* was entirely functional but lacked modularity. Routing logic, controller logic, and middleware were all tightly coupled—living inside shared files like trips.js and authentication.js. For example, the updateTrip function managed user authentication, input processing, database operations, and response handling all in one place. This made the codebase hard to maintain and difficult to scale.

There were no clear abstractions, little to no reuse of logic, and next to no documentation. API routes were all defined in a single index.js file, which created a bottleneck for scalability. Logging was handled with raw console.log() statements, and configuration values were hardcoded.

The planned enhancements—such as separating route logic into individual modules, introducing a middleware folder for reusable validators and authentication, and replacing all logging with Winston—directly align with the outcome of designing, developing, and delivering professional-quality software that is modular, maintainable, and production-ready. My planned use of JSdoc for documentation also supports the outcome of professional communication, ensuring the code is understandable to other developers.

Algorithms and Data Structures: Existing Functionality & Outcome Alignment

From an algorithmic standpoint, the project didn’t rely on advanced algorithm design, but the logic flows were very much present. Functions like register, login, and addTrip were functional but overloaded. Each performed multiple responsibilities—validation, logic, response formatting—without being broken into subroutines. There was also repeated logic throughout the app, such as checking for required fields or formatting responses, with no shared utility functions to reduce redundancy.

The application also lacked any kind of reusable validation logic. For example, field checks in register were repeated almost identically in login, and error handling was inconsistent between files.

My planned enhancements—extracting helpers like validateUserFields, introducing sanitization utilities, and refactoring route handlers into smaller, purposeful functions—show my ability to design and evaluate computing solutions using algorithmic principles. These changes reflect the outcome of structuring logic for maintainability and efficiency, and improving how the system handles input across varying edge cases.

Databases: Existing Functionality & Outcome Alignment

On the database side, the app used MongoDB with Mongoose to store user and trip data. While the basic schemas were in place and fields were marked as required, there was a noticeable absence of input validation, schema-level constraints (like min/max values), or indexing. Most controller functions passed raw req.body input straight to the database without sanitation or type enforcement. Queries were unindexed—like searching trips by code—causing full collection scans.

There was also no role-based access control. Anyone with a token could hit any endpoint, even update trips. There were no graceful shutdown handlers for the database, no centralized connection logic, and error handling in database interactions was inconsistent or missing altogether.

The planned database enhancements—refactoring schema fields, enforcing stronger validation and constraints, indexing query fields, and integrating environment-based config management—directly align with the outcome of developing a security mindset and using well-founded and innovative tools in software development. This also supports the broader goal of ensuring data privacy, stability, and performance in real-world backend systems.

Final Thoughts

While my original walkthrough touched on all these areas, this addendum clarifies exactly how each piece functioned before enhancement, and how my planned updates support the course outcomes. Taken together, the software structure, logic design, and data modeling of *Travlr* now reflect my growth in building production-grade systems—rooted in security, maintainability, and clear communication.