Austin Antles

CS 499

July 19 2025

### Artifact Enhancement Narrative: Software Design and Engineering

#### **1. Artifact Description**

The original artifact I am enhancing is a native Android application for inventory management, which I created for my CS 360: Mobile Architecture and Programming course. The application was developed using Java and Android's native UI toolkit with XML layouts. It functions as a self-contained, single-user tool, where all application components —user interface, business logic, and data storage via a local SQLite database —are bundled together in a monolithic architecture. It allows a user to create an account, log in, and perform all four CRUD (Create, Read, Update, Delete) operations on inventory items stored directly on the device.

#### **2. Justification for Inclusion and Enhancement**

I selected this artifact for my ePortfolio because it serves as a perfect "before" picture, representing a familiar and well-understood software architecture (the monolith) that is functional but has apparent, demonstrable limitations. Its primary weakness is its structure; being a self-contained unit, it cannot support multiple users, cannot be accessed from different platforms (like the web), and is not scalable. This makes it an ideal candidate for a significant and practical enhancement that showcases a wide range of modern software engineering skills.

The specific components that showcase my skills and abilities are found in the enhancement itself. The artifact was improved by completely re-engineering its backend architecture. I have decomposed the monolith into a modern client-server model by creating a robust, standalone backend service using the Go programming language (Golang). This new API now serves as the central authority for all business logic and data manipulation. The Go code demonstrates my ability to:

* Structure a professional web service with a clear separation of concerns (main entry point, database logic, request handlers).
* Handle HTTP requests and responses, process JSON data, and interact with a database securely and efficiently.
* Implement a clean RESTful API with logical endpoints for all CRUD operations.

This enhancement demonstrates my ability to analyze an existing system, identify its architectural flaws, and design a superior solution utilizing industry-standard practices and modern tools.

#### **3. Course Outcome Alignment**

This enhancement directly aligns with the course outcomes I planned to achieve in Module One. There are no updates to my outcome-coverage plan.

* **Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices.** My enhancement is a direct demonstration of this outcome. I evaluated the original monolithic design and identified its trade-offs (simplicity vs. lack of scalability). I then designed a new client-server solution that aligns with current computer science practices and standards. This new architecture addresses the original problem of inventory management in a scalable, accessible, and secure manner for multiple users.
* **Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices to implement computer solutions that deliver value and accomplish industry-specific goals:** By choosing to build the backend in Go, I am using a well-founded and modern tool that is highly sought after in the tech industry for its performance and concurrency. The technique of building a RESTful API to serve a decoupled frontend is the foundation of most modern web services. This enhancement delivers immense value by transforming a single-user mobile utility into a platform-agnostic service capable of supporting a real business.

#### **4. Reflection on the Process**

The process of enhancing this artifact was an incredibly valuable learning experience that forced me to shift my entire development perspective. The most significant challenge was transitioning from the mindset of a mobile developer, where everything exists in a single, contained environment, to that of a backend developer, where you are building a stateless service that must be secure, efficient, and reliable. Learning to structure a Go application, managing dependencies with go mod, and setting up a router for handling API endpoints was a steep but rewarding learning curve.

What I learned most profoundly was the importance of defining a clear API contract before writing the bulk of the logic. Deciding precisely what each endpoint would be (/api/v1/items, /api/v1/items/{id), what HTTP method it would use (GET, POST, PUT), and what the JSON structure for requests and responses would look like was a critical planning step. This made the actual coding of the handler functions much more straightforward because I had a clear blueprint to follow. Another major challenge was correctly setting up the PostgreSQL database and managing user permissions. I faced several "permission denied" errors. Overcoming this taught me that database security isn't just about the application code; it extends to proper administration of the database server itself. This process solidified my understanding of how different parts of a complex software system communicate and has given me a much stronger foundation in software architecture and design that I will carry with me into my career.