The artifact I chose to enhance for this part of my ePortfolio is the Go reverse proxy I originally built earlier in the year. It started as a simple routing layer that forwarded requests to two backend services, but over time it grew into something I used to explore more complex system behaviors. For this enhancement, I added two new subsystems: a health check module and a circuit breaker. These improvements are reflected in the enhanced files like main.go, app.go, and the new health.go and circuitbreaker.go additions. Even though this project wasn't tied to any class when I started it, it naturally became the strongest representation of what I've learned throughout my degree.

I selected this artifact because it shows my skills more clearly than any other project I've done. The original design already included things like routing logic, dynamic registration of backend services, caching, rate limiting, and basic load balancing. Enhancing it with a proper health check subsystem and a circuit breaker made it even more aligned with real-world software engineering practices. These additions reflect my understanding of system reliability, fault tolerance, and defensive design. They show I can take an existing system, identify what would make it more resilient, and implement those improvements without breaking the architecture. The enhancements also highlight my ability to use Go's concurrency features, organize code cleanly across packages, and think about behavior at the system level instead of just the code level.

For this enhancement, I planned to meet the program outcomes around software engineering, problem solving, and security. I feel like I met those goals. Adding health checks and a circuit breaker forced me to think more intentionally about how a system behaves under failure and what patterns help protect it. This ties directly into designing and evaluating computing solutions using sound algorithms and practices. It also relates to having a security mindset, because both features help reduce the blast radius when something goes wrong, whether that's due to a failure, an overload, or a potential exploit. I don't have any major changes to the outcomes I planned to cover. The enhancement lined up exactly with what I set out to demonstrate.

The biggest thing I learned while enhancing this artifact was how much planning matters when working on a system at this scale. The actual implementation itself wasn't difficult; what took the most time was thinking through how the pieces should fit together, how they should interact, and how to introduce new behaviors without creating unnecessary complexity. I had to slow down and make decisions about structure, boundaries, and communication between components. That ended up being more challenging than writing the code. I also learned how valuable it is to work iteratively; adding one subsystem at a time, testing it in isolation, and then plugging it into the main application.