Alexander Flood

SNHU CS-499

21st September 2025

**Artifact One Narrative**

For this artifact, I enhanced my existing Weight Tracker mobile application that was submitted as my Software Design and Engineering artifact. The Weight Tracker began as a standalone Android application in June 2025. The original version allowed users to register and log in, record weight entries, set up goals, and receive SMS notifications when they reach their goals. The app stored all data locally in SQLite, including plaintext passwords, and had no concept of multi-device persistence or secure authentication.

For this milestone, I refactored the project into a full-stack application by introducing a containerized REST API backed by a MySQL database. I also enhanced and secured the registration and login process by integrating Keycloak, an opensource and industry standard identity and access management solution. The Android app now communicates with the API using an OkHttp client and an Auth interceptor that retrieves and refreshes access tokens. Sensitive tokens are encrypted and stored locally using the Android Keystore for added security.

**Why This Artifact**

I chose this artifact because it demonstrates end-to-end software design decisions beyond UI coding. It demonstrates abilities and considerations around security, data modeling, API design, containerized deployment, and client-server integration. These are the skills most requested in full-stack roles from my own cursory job search. The enhanced artifact shows progression from a single-device demo, towards more professional and scalable architecture.

**Improvements**

The most significant improvements were in the areas of security, architecture, data persistence, deployability, networking, and user experience. On the security side, I eliminated the plaintext storage of passwords through integrating Keycloak, which provides secure authentication. Instead of the app storing usernames and passwords locally, users now authenticate through a secure identity provider and receive an access and refresh token which is managed by the application. This not only made the application safer, but it also aligns more closely with industry standards.

From an architectural point of view, I introduced a RESTful service layer using Node and Express, which cleanly separates the mobile client from the backend logic. All user data is now scoped using JWT (JSON Web Token) claims from Keycloak, ensuring that requests are properly isolated and prevent data leaks between users. This greatly improves the client-side architecture the original application was based around by introducing server-side ownership of the critical data-layer.

For the data layer, I migrated from a local SQLite instance to a MySQL database that will be hosted on a cloud provider such as AWS or virtual private server from a provider such as Digital Ocean. Weight entries and goal records are now stored in robust relational tables with server-side validation and ordering. A simple migration layer ensures that the schema is initialized consistently each time the API container starts, which improves stability and maintainability.

The deployability of the application greatly increased by containerizing the RESTful API with Docker Compose. The API, database, Keycloak, and reverse proxy (Caddy) run as separate services that can be consistently started and tested on any system. This makes the project more portable, easier to configure, test, and develop for other users on other machines and platforms. This brings the application closer to how production deployments are handled in many real-world environments.

Finally, I centralized all HTTP networking logic in a typed API client built with OkHttp in the Android app. An interceptor automatically attaches and refreshes authentication tokens for each request, which keeps the code in the activity classes clean and testable. I also broke down files in the Android application into sub-packages for added organization and clarity. Overall, the app continues to feel familiar to the end user, but under the hood, it has been transformed into a much more professional and scalable system.

**Outcomes**

These enhancements demonstrate how I made progress towards outcomes 2, 4, and 5. For outcome 2, I documented the new architecture, API surface, deployment services, and security decisions in ways that communicates clearly to fellow developers. For outcome 4, I designed and implemented a modular client-service database architecture, containerized the system with Docker, enforced per-user scoping, and introduced simple migrations and health checks, which together reflect better software and engineering practices. For outcome 5, I eliminated plaintext password storage entirely, adopted secure authentication via Keycloak, enforced token validation server-side, and encrypted tokens for local storage. These changes not only minimized the risk of exposing sensitive data, but it also demonstrated a security-based mindset around the total project.

**Planned vs Achieved**

In module one, I planned to target outcome 4 (software engineering/design), outcome 5 (security mindset), and outcome 2 (professional communication). With this milestone, I believe I met these targets. I delivered a containerized service, redesigned the data-layer, implemented secure authentication, and provided clear documentation.

There are other outcomes I hope to achieve such as outcome 1 (collaboration) and outcome 3 (algorithmic design), but they are less central here and will be addressed in future iterations.

**Reflection**

The refactor required deciding how to balance the needs of the user interface with the service constraints. I designed the API first and then realigned my Android client to ensure it fit the API as needed. I introduced per-user scoping via a JWT, which provides each user with a unique identifier when decrypted and enforces proper isolation of data. I also adopted a robust migration step that runs whenever my web services boot up, ensuring that MySQL always reflects the correct schema.

The biggest shift was treating authentication as a primary concern. Instead of storing passwords in SQLite as plaintext, the client now performs a modern authentication flow to obtain secure access and refresh tokens. These tokens are refreshed automatically and encrypted before being stored locally using the Android Keystore, which is a major improvement in terms of both security standards and user trust. This change also forced me to think about how security affects all parts of my design.

Through this enhancement, I improved how I design and engineer my systems and applications with separation of concerns in mind. By decoupling the client from the server and giving each a responsibility, the code became easier to test, extend, and maintain. I also gained hands-on experience with containerized deployments and integrating industry-standard open-source software, which deepens my understanding how software is built in a professional environment.

**Challenges**

A key challenge was managing token lifecycles without disrupting the user experience. I solved this by coordinating the refresh logic within an OkHttp interceptor and persisting the updated authentication state after redirecting from a successful login. Data correctness was another issue, as string-based dates caused ordering problems. I solved this by migrating to TIMESTAMP columns and enforcing descending order on the server side of the application. Ensuring per-user isolation on the data required deriving ownership from the JWT and scoping all app queries by providing the user’s unique access token. Finally, I faced some initial challenges getting my Android virtual device to trust my development SSL certificates, which prevented me from testing my application. I fixed this by trusting my development SSL certificates, which will not be an issue in a production environment where I will have official certificates that do not require this extra step.

**Future Work**

Planned enhancements include adding an offline-first encrypted SQLite cache and discrete background synchronization. Algorithmic enhancements include introducing simple projections for when weight goals will be met as well as rolling averages to smooth out noise in any weight entry trends. Finally, I will introduce a production grade Docker Compose file that is configured for running in a secure production environment. Currently, only a development grade Docker Compose file exists inside the project.

**Conclusion**

This milestone elevates the Weight Tracker from a local demo to a more professional, secure, and deployable full-stack application. The artifact now shows practical software design, modern tooling, and secure practices that closely align with industry expectations while complimenting my ePortfolio and fulfilling its objectives.