Nicholas Pellegrini

CS-499 Computer Science Capstone

3-2: Software Design and Engineering Enhancement Narrative

7/20/2025

For this milestone, I focused on enhancing my original weight tracking Android app by improving its architecture, data handling, and overall user experience. The goal was to refactor the app so it would be easier to maintain, test, and scale. I wanted the app to feel more professional and modern, and to meet the expectations of good software design practices. To do that, I transitioned the app from a simple activity-based structure into one that uses the MVVM (Model-View-ViewModel) architecture, with Room as the new persistence layer.

Originally, the app relied on a helper class that used raw SQLite queries. This made the code harder to read and maintain. The UI logic was mixed in with the data access logic, which meant there wasn’t a clean separation of concerns. I also noticed that user input wasn’t being validated well, and the app had limited responsiveness when data changed. It worked, but it wasn’t scalable, and updates would have been risky and time-consuming to implement.

I started the enhancement by designing a clear separation between the UI, the ViewModel, and the database layer. I created a WeightEntry model class to define the structure of the data, then implemented a WeightDao interface to handle database operations using Room. I set up AppDatabase as a singleton Room database, so the app can safely and efficiently access stored weight entries. I added a WeightRepository to act as the middleman between the ViewModel and the database, and it uses a background thread to avoid blocking the UI.

With the data layer in place, I rewrote the WeightViewModel to expose the list of weight entries as LiveData. It also handles all user input validation. If a user enters an empty string, a non-numeric value, or a weight that doesn’t make sense, the ViewModel posts an error message that the UI can respond to. This helped move all the business logic out of the activity and made the app much cleaner.

In the UI, I replaced the old adapter with a modern ListAdapter that uses a diffing strategy, so updates to the list are more efficient. The activity observes the ViewModel’s LiveData and updates the RecyclerView when changes occur. I also added a Snackbar to show error messages whenever input is invalid. These changes made the app feel more responsive and gave better feedback to the user.

One feature I wanted to include was the ability for users to switch between light and dark mode. I added a toggle switch in the UI and a ThemePrefs class that stores the user’s choice in shared preferences. When the app starts, it checks that preference and applies the correct theme. Toggling the switch saves the preference and recreates the activity, so the theme applies immediately. This made the app more customizable and user-friendly.

Testing went smoothly. I tested input validation, theme switching, data entry, and rotation behavior. Everything worked as expected, and the app preserved state and layout correctly. I didn’t hit any major issues once the Room dependencies were added and Gradle was synced.

Looking ahead, I’d like to move the theme toggle and the SMS feature into a proper settings screen. Right now, they’re functional but not in the most intuitive place. I’d also like to refactor the login screen (MainActivity) and the SMS permission screen to use the same ViewModel-based structure I used here. That would make the whole app more consistent and easier to test.

Overall, this enhancement dramatically improved the app. It’s now using industry-standard architecture, the code is modular and easier to understand, and users get real-time feedback when they interact with the app. This project gave me a chance to demonstrate my ability to design and implement a clean software architecture, validate user input securely, and enhance the user experience with persistent theming. It also shows I can work with modern tools and libraries like Room and LiveData, and structure an app in a way that aligns with professional software development practices.