Austin Antles

CS 499

Aug 3 2025

### Artifact Enhancement Narrative: Databases

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

The original artifact I am enhancing is the database component of my Android inventory management application, which was created for my CS 360: Mobile Architecture and Programming course. A local, file-based SQLite database handled the application's data persistence. This database was embedded within the application itself, meaning each installation had its own isolated data file. The schema was simple, and critically, it stored user passwords in plain text, representing a significant security flaw. This approach, while functional for a single-user offline prototype, is fundamentally unsuitable for a real-world system that requires data integrity, security, and shared access.

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

I selected this artifact because its database implementation provides a clear and compelling opportunity to demonstrate growth in two critical areas: database architecture and security. The migration from SQLite to a server-based system showcases my understanding of scalable data solutions, but the most important enhancement is the implementation of proper security practices.

The artifact was improved by migrating the data layer to **PostgreSQL** and, more significantly, by implementing a **secure user authentication system**. The specific components that showcase my skills and abilities are:

* **Database Schema Enhancement:** I designed and created a new users table in PostgreSQL with appropriate constraints, including a UNIQUE constraint on the username to ensure data integrity.
* **Secure Password Storage:** The core of this enhancement is in the auth.go and auth\_handlers.go files. I implemented functions to securely hash user passwords using the industry-standard **bcrypt** algorithm before storing them. The HashPassword function creates a secure hash, and the CheckPasswordHash function provides a safe way to verify a login attempt without ever exposing the original password. This demonstrates a proactive security mindset.
* **Authentication Logic:** I built API endpoints for user registration and login that correctly handle the flow of data, from validating user input to interacting with the database and securely comparing password hashes.

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

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

* **Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources.** This outcome is the primary focus of my enhancement. I identified the critical vulnerability of storing plain-text passwords in the original artifact. I then mitigated this design flaw completely by implementing bcrypt hashing. This directly anticipates a common adversarial exploit (a database breach) and ensures that even if the data were compromised, user passwords would remain secure and private, thus protecting user data.

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

The process of implementing this enhancement was incredibly insightful and solidified my understanding of backend security. The biggest challenge was not just learning how to use the bcrypt library, but understanding *why* it is so important. It forced me to think like an adversary and consider the potential consequences of a data breach. This shifted my perspective from just "making login work" to "building a secure authentication system."

One of the key things I learned was the practical difference between encoding, encryption, and one-way hashing. I now understand that hashing is the correct tool for password security because you should never need to decrypt a password, only verify it. I also faced the challenge of structuring the new authentication code in a clean and efficient manner. My solution was to create separate auth.go and auth\_handlers.go files. This separated the pure cryptographic utility functions (such as HashPassword) from the HTTP request handling logic (such as RegisterHandler). This separation of concerns made the code much cleaner, easier to read, and more maintainable. This process has fundamentally changed how I will approach any feature involving user data in the future; security will no longer be an afterthought but a foundational requirement from the very beginning of the design process.