**Professional Self-Assessment**

Joshua Brown

Department of Computer Science, Southern New Hampshire University

CS – 499: Computer Science Capstone

Mike Alesso, M.Sc.

December 14, 2024

**Professional Self-Assessment**

I have learned much and gained many skills from this program and have been given the opportunity to culminate a bit of everything I have learned into a final project that showcases some of the key concepts that I have picked up along the way. It is through the completion of course work that I have learned vital principles that have helped me shape my values and work ethics as a software developer. Through the development of a secure coding standard and security policies I have learned the need to implement security throughout the lifecycle of an application which will help maintain defense-in-depth (a multilayered defense), and how preventing the spread of errors in the application by addressing them at the most basic level (development) will make a stronger, more secure application, with fewer holes for hackers to attempt entry. Through courses like reverse engineering, I’ve learned the importance of understanding assembly language when it comes to maintaining and updating legacy code, and how documenting and commenting code upon updating, can help further increase the overall efficiency and collaboration between teams, despite there being a vast number of moving parts. I’ve even learned the roles of key figures, their responsibilities and how each phase impacts the software development lifecycle.

For my final capstone, I chose to do one all-encompassing project that includes all three requirements: software engineering, algorithms & data structures, and database. The original artifact was from my Mobile Architecture & Programming Class, where I chose to create an inventory app that allowed a user to create and track inventory items, as well as be notified when an item is out-of-stock. This was our first project that utilized all three of the elements, and although I thought that it was done well after its completion, I didn’t realize until later just how flawed the original project was, which I outline below.

* **Original Artifact Flaws:**
  + **Unsecure login:** Though the application required users to login or create an account if they did not have one, the user did not have a unique inventory experience. This means that if a user creates a new account on the same device as another user, then the user could essentially gain access to the previous user’s inventory even after they logged out.
  + **Unencrypted data:** Though the application made use of databases, for the user and inventory, the data stored in the databases was unencrypted, so even if the first issue was fixed, a hacker could still access unencrypted information in the event that a user’s account got hacked.
  + **All datasets not manageable:** Though I fully implemented CRUD functionality for the inventory items, I did not fully implement CRUD functionality for users; in particular, I did not finish Update (the ability for a user to update their account) and Delete functionality (the ability for a user to delete their account).
  + **Hardcoded Variables:** Lastly, I left a hardcoded variable in the code that served as a key feature in the application, which was the phone number for the SMS feature that notifies the user if an item is low or out of stock.
* **Artifact Enhancements addressing the issues:**
  + **Unsecure login:** In order to solve this issue, I realized that I would have to in some way tie the user to the items that they created in the database. This meant that I would have to modify the item’s table to include a column that would have the user’s email address, so that when a user logged in, they would only be able to pull items belonging to that user. I also had to make keeping up with items more manageable, so I changed the primary key to be a composite key consisting of the creator’s email along with an item’s id number so items could be efficiently pulled from the database when needed.
  + **Unencrypted Data:** To fix this, I started by adding a hashed password and salt column to the user database. I then created an algorithm to hash a password and generate a random unique salt for users. This hashed password and salt combination would be used as unique keys to encrypt and decrypt data. I then created functions to encrypt the data upon adding or updating the data into the database, and between transferring the data between transition points in the app. Lastly, I created functions to decrypt the data when I needed to display the data to the screen.
  + **All datasets not manageable:** For this issue, I added/updated user CRUD methods to the user and item repositories, I then added a user account screen to the mobile app that allowed a user to view their account details, update, or delete their account using an interface. I then added further functions that implemented inventory updates on all of a user’s items when a user changed the details of their account, or wipe all of a user’s items when a user initiated a delete of their account.
  + **Hardcoded Variables:** To fix the hardcoded phone number issue, I added a phone number column to the user’s table, and added a way for a user to add/edit their phone number from their account screen. I then removed the hardcoded variable, and instead used the user’s phone number on file (if they had one). Since I did this, I had to make the SMS feature an optional feature, that only implemented if a user had a phone number on file.

**Conclusion:**

With the addition of all of the enhancements, I now have an application that allows a user to log into their account (or create a new account) and have access to their own individual items. With the implementation of hashed password and random unique salt generation I have made the log in process more secure and less likely to succumb to hacking via brute force attacks. With the use of encryption methods at rest, in transit, and in use I have further increased the security of the application, so that in the event that a breach does occur, my users’ data will still be safe, due to the hackers not being able to access readable or hardcoded data. Lastly, users can freely make changes to their accounts, without losing access to their stored data, and users can take confidence in the fact that if they delete their account, all of their data relating to their account will be deleted as well. This is an application that not only showcases my growth as a software developer, but it also displays my ingenuity in solving problems, and my ability to write secure and reliable code that truly shows defense in depth, ensuring a trustworthy and seamless experience for all users.