Secure Password Manager

Isaac Kydd

July 31 2025

Software Testing and Validation (ENSE 375)

University of Regina | Go far, together.

Agenda

- Introduction
- Problem Definition
- Design Requirements
- Solutions
- Testing and Demonstration
- Project Management
- Conclusion and Future Scope

Introduction

- I struggle to remember passwords I make for accounts
- Have to constantly reset passwords, sometimes this process takes a lot of time depending on the website and can be a big problem
- Need a way to store passwords, but not in a way where they can be stolen

Problem Definition

The problem that I am trying to solve is that I do not want to write down passwords in plain text, either on a computer or in real life, as they could potentially be stolen. However, since I cannot remember all of them, especially the ones I use less frequently, I want away to recall them when I need them.

This will require a method where I can store the passwords, but not be at risk of having them stolen.

Functions

- Store passwords securely in an encrypted format.
- Retrieve passwords for the user after secure authentication.
- Authenticate users through a secure master password or other secure methods.
- Validate password strength based on length, complexity, and uniqueness criteria.
- Delete stored credentials securely upon user request.
- Import/Export password data in a secure manner for backup or transfer.
- Test internal logic and functions using unit, integration, and system testing strategies.

Objectives

- Secure - to protect user data from unauthorized access.
- Reliable - to function consistently under different conditions.
- User-friendly - to be easily navigable and understandable by users with minimal training.
- Efficient - to operate with minimal resource usage while maintaining performance.
- Maintainable - to allow future modifications or updates with ease.
- Testable - to support thorough verification through various testing methods.
- Portable - to work across multiple platforms or systems if needed. Design Requirements
- Private - to ensure the user’s privacy and data rights are respected.

Constraints

- The password storage must be encrypted using industry-standard algorithms
- The application must authenticate users before allowing access to stored passwords.
- The application must operate within a local system environment (no cloud dependency).
- The design must follow test-driven development (TDD) methodology.
- All four design constraints (economic factors, security compliance, reliability, societal impact) must be addressed.
- The entire application must be developed and tested within two months.

Solution 1

Solution 1 was greatly flawed. It involved storing user information in a JSON file (encrypted of course), to make it easier to look at. This however had a ton of scaling and organizational issues, and was quickly abandoned for a more formal database approach.

Solution 2

Solution 2 was much more similar to the final design.

- MySQL for data storage
- Java for logic
- Users in one database, user information in other database

Solution 3

- Uses SQLite to create database from within java
- Uses JDBC to interface with database
- Use 1 table for user logins (to program)
- Create unique table for each user to store their credentials

This solution was chosen because it was much more testable.

Everything being in java meant that the entire program could be tested with the same programs.

Test Requirements

- Can create unique hash for every user's master password
- Can encrypt user data using master password
- Can decrypt user data to original input (only if master password is given)
- Can store data into database and then retrieve it accurately
- Prevent null inputs
- User can only see data for their credentials
- User can register, login, and enter information into their data table
- User information persists through sessions
- Goal of 80% code coverage, although this may differ depending on how UI-heavy code is (difficult to test if a blank space padding the UI is rendered right with JUnit)

Test Cases

- Extensive unit testing on encryption java class
- Extensive unit testing on database manager java class
- Integration testing with logic controller class

  Register, Login, Add Credentials, Delete credentials, etc
- Complete system testing with UI

- For UI testing, manual user testing may be required to ensure accuracy (will be difficult to automatically test if button is rendered in correct spot, would be easier to test manually anyways)

Results

- Data was able to be reliably encrypted and decrypted
- Data was able to be reliably stored and retrieved upon request
- Logic controller was confirmed to be able to register a user, login to an existing profile, and enter information that would be stored and then retrieved from the database
- UI testing with automated JUnit tests has limited success

- Direct user testing was much more useful for testing UI elements

Program Demonstration

Conclusion and Future Work