Software Implementation and Testing Document

For

Group 14

Version 1.0

Authors:
Samantha Bui
Ludginie Dorval
Antonio Garriga
William Lee
Lillian Malik

1. Programming Languages (5 points)

1. **Python**: Employed in both back-end and front-end development, as well as for database integration. Python was chosen due to its versatility, extensive library support, and ease of use. Specifically, of Python Flask and Encryption libraries.
2. **SQL**: Utilized for managing and interacting with the back-end database. SQL was selected for its reliability and its ability to handle database management.
3. **HTML**: Used for structuring the front-end user interface. HTML was chosen because it is the standard language for web page design. In future increments we will also be using CSS to enhance the visual aspect of the project.
4. **JSON**: Used for data formatting and communication between the back-end and front-end components. JSON was selected for its simplicity and efficiency in representing complex data structures, as well as its compatibility across different languages.

2. Platforms, APIs, Databases, and other technologies used (5 points)

1. **MySQL**: Implemented as the back-end database management system.
2. **Flask**: Utilized for the development of the front-end framework and for facilitating secure database interactions. Flask was selected for its versatility in integration.
3. **Cipher**: Integrated for front-end encryption and secure data handling. Cipher was chosen due to its strong encryption capabilities.
4. **JSON**: Used for data formatting and communication between the back-end and front-end components.
5. **Virtual Studio Code**: We used this Integrated Development Environment (IDE) for writing, debugging, and running code.
6. **GitHub:** Used for team collaboration, providing easy access to source code and documentation.

3. Execution-based Functional Testing (10 points)

1. Cost-Volume-Profit (CVP) Analysis: By testing and verifying calculations.
2. Production Budgeting: Tested budgeting formulas to ensure correct inventory predictions.
3. Role-Based Access Control (RBAC): Ensured proper user access based on assigned roles and restricted unauthorized actions. Tested by running Flask and logging in with initialized admin, user, viewer. The front-end logic was tested independently from the back-end logic. As of Increment I, we have not fully integrated the front-end logic with the back-end logic. However, as we progress, we will integrate efficiently to implement (RBAC) and Encryption.
4. Encryption: The intention was to confirm that passwords and sensitive user data are securely encrypted before storage. However, we focus on other areas of logic. The front-end logic was tested independently from the back-end logic. As of Increment I, we have not fully integrated the front-end logic with the back-end logic. However, as we progress, we will be sure to integrate efficiently, so that we can fully implement (RBAC) and Encryption.

4. Execution-based Non-Functional Testing (10 points)

Due to unforeseen challenges encountered during the process of merging the front-end and back-end components, we were unable to conduct comprehensive testing of the non-functional requirements within this increment. These complications have delayed our ability to fully assess critical system aspects such as performance, security, and scalability, which are essential for ensuring the robustness and reliability of the application.

5. Non-Execution-based Testing (10 points)

We conducted bi-monthly meetings to review our coding progress and establish objectives to be achieved by the subsequent meeting. The meeting notes for each session are documented in the GitHub README file of the Increment 1 repository. During these meetings, with a majority of team members present, we engaged in discussions regarding the code and the recent modifications made to the repository.