Student No.: 2102675

**Functional Requirements:**

1. Asset Management:

* Users should be able to add new assets to the system and should include the ability to add details such as system name, model, manufacturer, type and Ip address, an optional text note for the systems with physical stickers marking their purchase date or for extra information should also be available.
* Users should be able to view all asset details, this should include data fields such as the assets unique identifier (ID), system name, model manufacturer, type, Ip address, purchase date and text notes (if applicable).
* Users should be able to edit the assets and their fields.
* Users should be able to delete assets from the system.
* Each asset should have a unique identifier.

1. Automatic Data Acquisition:

* The software should automatically grab system info upon first boot on a given system.

1. Employee Management:

* Users should be able to add employees responsible for assets, this should include their first and last name and their email address.
* Users should be able to edit employee information.
* Users should be able to delete employee information.
* Users should be able to track which assets an employee has responsibility over.
* Employees should be assigned to a department on creation.

1. Department Management:

* Predefined departments built into the system, given extra time the option to add, edit, delete and merge should be available.
* Each department should have a list of employees.

1. Search and Filter:

* Users should be able to search and filter for departments, employees and assets by name, model, Ip address and unique identifiers.

1. Database Integration:

* All assets, employees, and department information should be stored in an online database for future access and integration with other company systems.

**Non-Functional Requirements:**

1. Performance:

* The system should have a minimal response time when corresponding with the database.
* Screen Refreshes should not impact or cause delay to the user.
* The system should be able to manage a growing number of assets, employees and departments as the company expands, without significant degradation in performance.
* The system should ensure high availability with minimal downtime.
* The system must reflect all asset, employee, and departmental data consistently across the system and database.

1. Usability:

* The system should have a user-friendly interface that is intuitive and helps minimise user error.
* The system user training time should take no longer than 30 minutes.
* The system should require minimal technical knowledge to operate and navigate.
* Given enough time Localisation support should be added as Scotland and by proxy Scottish Glen have two national languages.

1. Security:

* The system must ensure that sensitive data such as employee email and asset information is stored securely.
* The system should include rudimentary user authentication to prevent accidental or uninformed access.
* The system should include some form of audit log, so all changes are available for accountability, legal and safety purposes.

1. Compatibility:

* The system needs to work with the Microsoft operating system.
* The system must use a widely used database that can be accessed by other systems in the future.

1. Maintainability:

* The system should be designed using well documented code to allow future maintenance, updates, and features additions if necessary.
* The system’s design should allow for future integration with other Scottish Glen company systems without requiring a major overhaul.

1. Data Backup And Recovery:

* The system should offer an option to backup database information however access to it should be guarded behind an additional level of authentication to prevent bad actors.
* The system should offer the option to recover information back onto the database again guarded by additional authentication.

**Agile Methodology:**

For the development of this system, an agile development approach is required, this is to allow for iterative development, continuous feedback, and improvement, and allows for the project to be adaptable to client requirements. Given the brief I believe that the agile methodology is the best one to pick as it matches the requirements I must meet and ensures I am meeting the client’s request for a responsive and evolving system that provides flexibility in handling changes throughout the process.

I have broken down the development into X sprints, each focusing on specific tasks and deliverables:

Sprint 1:

* Set up database schema for assets, employees, and departments.
* Establish basic project structure in my chosen language (It is between Java and C# right now I will decide after this).

Sprint 2:

* Implement functionality to add, edit, delete, and view assets.
* Automate asset data acquisition.

Sprint 3:

* Implement employee functionality to add, edit, delete, and view.
* Integrate static department information.

Sprint 4:

* Conduct unit and integration testing for database interactions and automatic data capture.
* Client feedback session to validate usability and functionality (Whether this is my flatmate pretending to be the client or feedback from the lecturer is remaining to be seen)

Sprint 5:

* Improve system based on feedback from client.
* Implement security features and complete a final unit test.

**Code, Dev Tools and Processes:**

This project will be developed using VSCode and Microsoft SQL Server primarily for readability while I code in VSCode’s case and because that is how I access the given database for the project in MSS’ case.

A blue logo with white text

Description automatically generated

Some simple I principles I plan to follow through development are:

* DRY (Don’t Repeat Yourself) By Abstracting common methods I will not be writing and rewriting the same code multiple times.
* Separation of Concerns, By splitting the UI, Communication logic, and basic functions of the program into separate classes I can increase readability and allow the program to be more maintainable over a longer period.

I plan to use Source Control during this project ensuring that changes can be view and rolled back, if necessary, this also allows for progress in the project to be tracked by the client should that be required.

**Design Pattern And Principles:**

* SRP (Single Responsibility Principle) Each class in the system has one clear responsibility.
* OCP (Open/Closed Principle): The system will be designed to allow extensions without modifying core components.
* Factory Pattern: Used for creating instances of assets and employees, ensuring that objects are consistently constructed with all fields populated.

I intend to adhere to these stipulations to construct a system that allows for maintainability and future ease of integration.

**Reliability, Secure Coding Practices, and Testing:**

Reliability:

* The system will be designed with reliability in mind by ensuring proper error handling and validation mechanisms are in place.
* Transactional database operations will be used to ensure data consistency.

Secure Coding Practices:

* All user input will be validated to ensure no injection attacks and that information is in a valid format.
* Advanced Authentication is not required how ever it will be implemented for any major actions e.g. deleting an asset.

Testing:

* Unit Tests will be conducted to ensure core functionality.
* Before final delivery the whole system will be stress tested to ensure reliability and performance.