Part 1 — *PROG7311*

Keagan Tomlinson

ST10084431
Varsity College

PROG7311

Nash Ramckurran

1/04/2023

Introduction

As the only developer responsible for creating Farm Central's mushroom stock management website, I have composed this report to address the non-functional requirements and design patterns for the project. Although I am working solo, my skills and experience make me confident in my ability to deliver a high-quality system that satisfies client needs and follows industry best practices.

This report aims to offer a comprehensive outline of my proposed solution for Farm Central's mushroom stock management website, including strategies to address non-functional requirements and the design and architectural patterns I will employ to accomplish my objectives. My proposal is informed by substantial experience in creating similar systems for other clients and a deep understanding of Farm Central's unique project requirements.

The report is organised into several sections, beginning with an analysis of the non-functional requirements I have deemed crucial for the project. I will then delve into my proposed solution in more detail, discussing the design and architectural patterns I intend to apply in developing the system. Additionally, I will address how I plan to test the system to ensure compliance with non-functional requirements and manage the project to guarantee timely completion.

As a solo developer, I am dedicated to delivering a system that caters to client needs and adheres to industry best practices and standards. I have closely collaborated with the client to guarantee that my proposed solution aligns with their specific requirements and that the system is both functional and user-friendly.

In conclusion, I believe my proposed solution is the optimal choice for Farm Central's mushroom stock management website project. I am confident in my ability to deliver a system that not only satisfies client needs but also complies with industry best practices and standards. I eagerly anticipate the opportunity to collaborate with Farm Central and provide a system that enhances their ability to achieve their goals and better serve their customers.

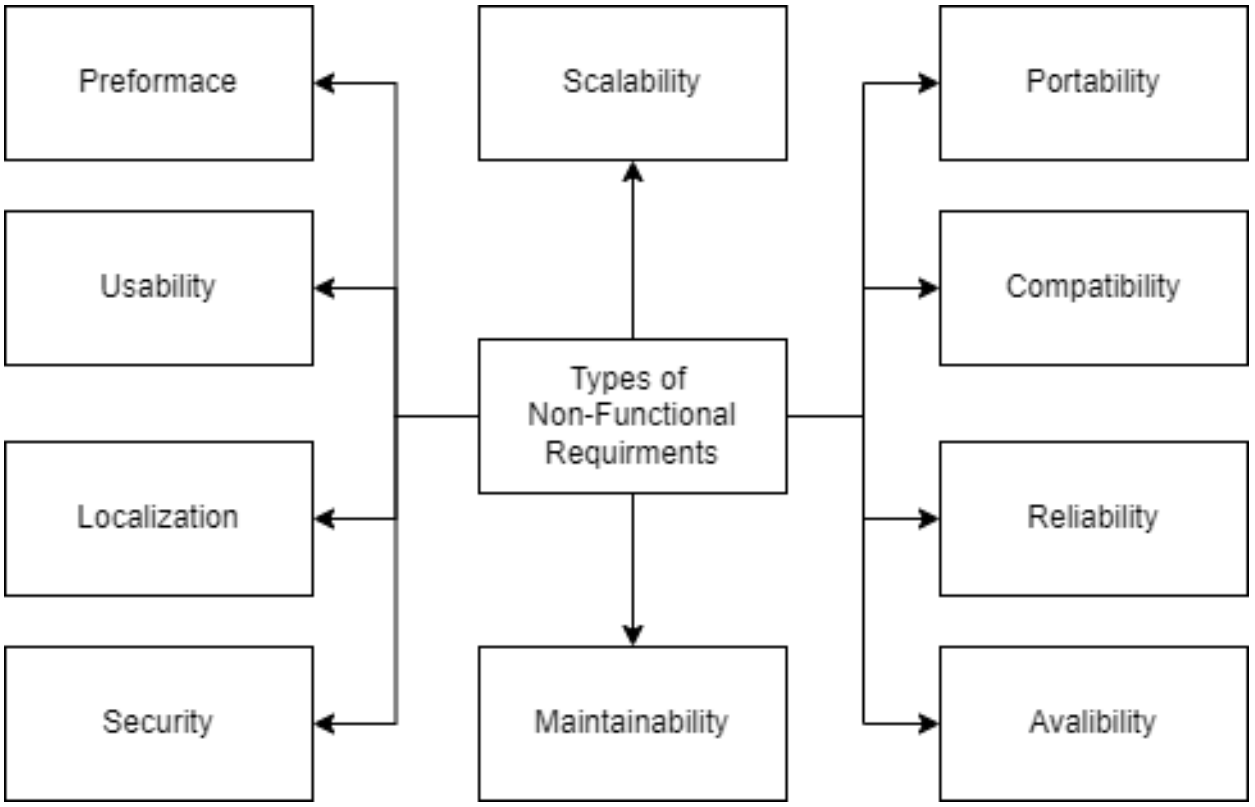
Non-Functional Requirements

The non-functional requirements of the Farm Central stock management website project are critical to ensuring that the system meets the needs of the client and its users. These requirements cover a range of areas, including security, performance, scalability, usability, compatibility, reliability, accessibility, and compliance. (AltexSoft, 2019)

- Security: Security is a top priority for the Farm Central stock management website project. The system will need to protect sensitive information such as inventory data, customer information, and financial data from unauthorised access, modification, or theft. To achieve this, the system will use a range of security measures, including authentication, authorization, encryption, and access controls. The system will also be regularly tested and audited to ensure that it meets the highest standards of security. (AltexSoft, 2019)
- Performance: The performance of the system is a critical factor in ensuring that it can handle the demands of Farm Central's business operations. The system will need to be able to handle a large volume of transactions, process requests quickly, and provide real-time data to users. To achieve this, the system will be optimised for speed and efficiency, with a focus on minimising latency and maximising throughput. The system will also be carefully monitored to ensure that it meets the performance requirements at all times.(AltexSoft, 2019)
- Scalability: The system will need to be scalable to accommodate the growth of Farm Central's business. As the number of users and transactions increases, the system should be able to handle the increased load without any degradation in performance. To achieve this, the system will be designed with scalability in mind, using techniques such as load balancing, horizontal scaling, and caching. (AltexSoft, 2019)
- Usability: The usability of the system is critical to ensuring that users can easily and efficiently perform their tasks. To achieve this, the system will be designed with a user-centred approach, with a focus on simplicity, clarity, and ease of use. User feedback will be incorporated throughout the development process to ensure that the system meets the needs and expectations of its users. (AltexSoft, 2019)
- Compatibility: The system will need to be compatible with a wide range of devices and platforms, including desktops, laptops, tablets, and smartphones. To achieve this, the system will be designed with a responsive user interface that adapts to different screen sizes and resolutions. The system will also be tested on a range of devices and platforms to ensure that it works consistently across all of them. (AltexSoft, 2019)
- Reliability: The reliability of the system is critical to ensuring that it can be relied upon to perform its tasks consistently and accurately. The system will be designed with a focus on fault tolerance, using techniques such as redundant servers, failover mechanisms, and data backups. The system will also be tested extensively to ensure that it is reliable under a range of conditions. (AltexSoft, 2019)
- Accessibility: The system will need to be accessible to users with disabilities, in compliance with accessibility standards such as WCAG 2.1. To achieve this, the system will be designed with accessibility in mind, using techniques such as semantic markup, keyboard navigation, and alternative text for images. The system will also be tested with assistive technologies such as screen readers to ensure that it is accessible to all users.(AltexSoft, 2019)

- Compliance: The system will need to comply with relevant laws and regulations, including data protection laws and industry standards such as PCI DSS. To achieve this, the system will be designed with compliance in mind, using techniques such as encryption, data anonymization, and access controls. The system will also be regularly audited to ensure that it meets the relevant compliance requirements. (AltexSoft, 2019)

Overall, the non-functional requirements are critical to ensuring that the Farm Central stock management website project is a success. By addressing each of these requirements in a comprehensive and systematic way, the system will be able to meet the needs of the client and its users and provide a secure, reliable, and efficient platform for managing inventory and supporting business operations.



Solid

The SOLID principles are a set of guidelines that help developers write code that is easier to maintain, extend, and modify. Let's take a closer look at each of these principles:

1. Single Responsibility Principle (SRP)

The SRP states that every class or module should have only one responsibility or reason to change. In other words, a class or module should do one thing and do it well. This makes the code easier to understand, test, and modify because each component has a clear and specific purpose.(Pentalog, 2022)

For example, if you have a class that handles both user authentication and email sending, it would violate the SRP. Instead, you should split these responsibilities into separate classes, one for authentication and one for email sending.(Pentalog, 2022)

2. Open/Closed Principle (OCP)

The OCP states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This means that you should be able to add new functionality to a system without having to modify existing code.(Pentalog, 2022)

For example, if you have a class that calculates the total price of an order, you should be able to add a new discount code without having to modify the existing code. You could achieve this by using a strategy pattern or a decorator pattern.

3. Liskov Substitution Principle (LSP)

The LSP states that subtypes (subclasses) should be substitutable for their base types (superclasses) without affecting the correctness of the program. This means that you should be able to use a subclass wherever the superclass is expected.(Pentalog, 2022)

For example, if you have a class that represents a shape, you should be able to substitute a subclass that represents a rectangle without changing the behaviour of the program.

4. Interface Segregation Principle (ISP)

The ISP states that clients should not be forced to depend on interfaces they do not use. Instead, interfaces should be fine-grained and focused on specific use cases.

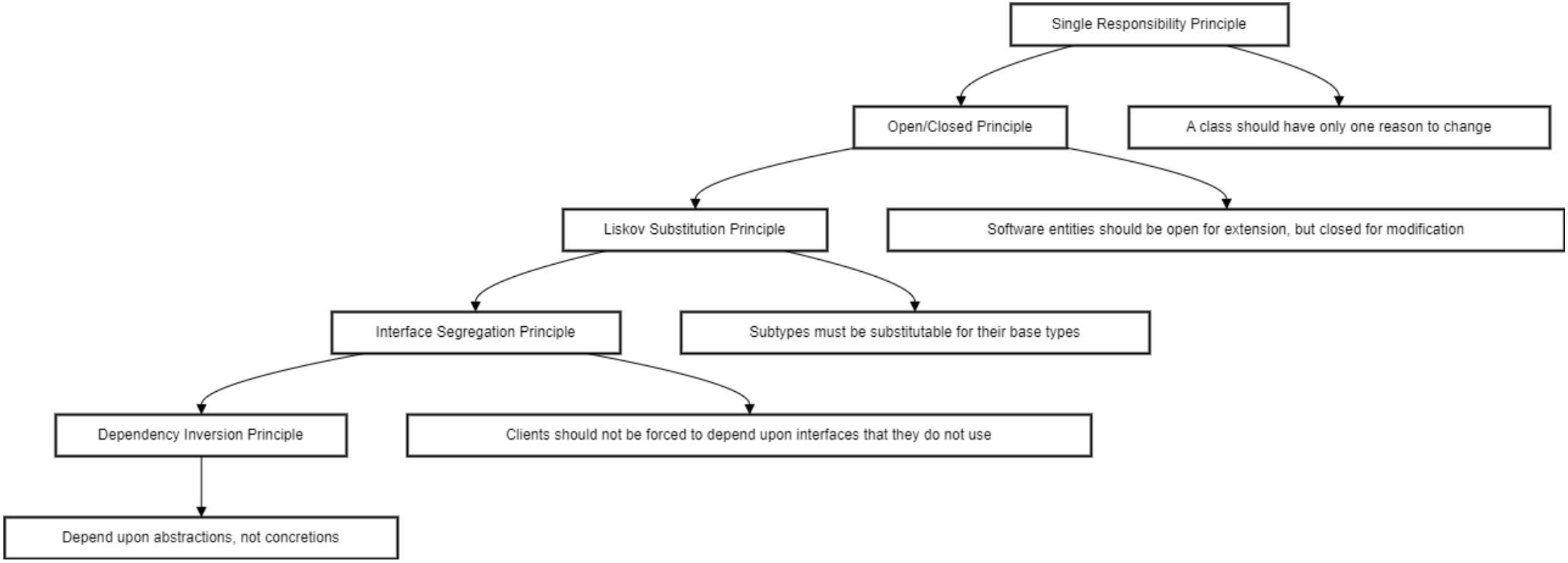
For example, if you have a class that implements a large interface with many methods, you should split that interface into smaller, more focused interfaces. This makes the code easier to understand and reduces the risk of breaking changes.(Pentalog, 2022)

5. Dependency Inversion Principle (DIP)

The DIP states that high-level modules should not depend on low-level modules. Both should depend on abstractions (interfaces or abstract classes). This means that you should depend on abstractions rather than concrete implementations.(Pentalog, 2022)

For example, if you have a class that needs to read data from a database, you should depend on an interface that abstracts the database access rather than a concrete implementation. This makes the code more flexible and easier to test.

Overall, the SOLID principles provide a set of guidelines that can help developers create software that is modular, flexible, and easy to maintain. By following these principles, developers can create software that is less prone to bugs, easier to understand, and more adaptable to changing requirements.(Pentalog, 2022)


Single Responsibility Principle
Open/Closed Principle
A class should have only one reason to change
Liskov Substitution Principle
Software entities should be open for extension, but closed for modification
Interface Segregation Principle
Subtypes must be substitutable for their base types
Dependency Inversion Principle
Clients should not be forced to depend upon interfaces that they do not use
Depend upon abstractions, not concretions

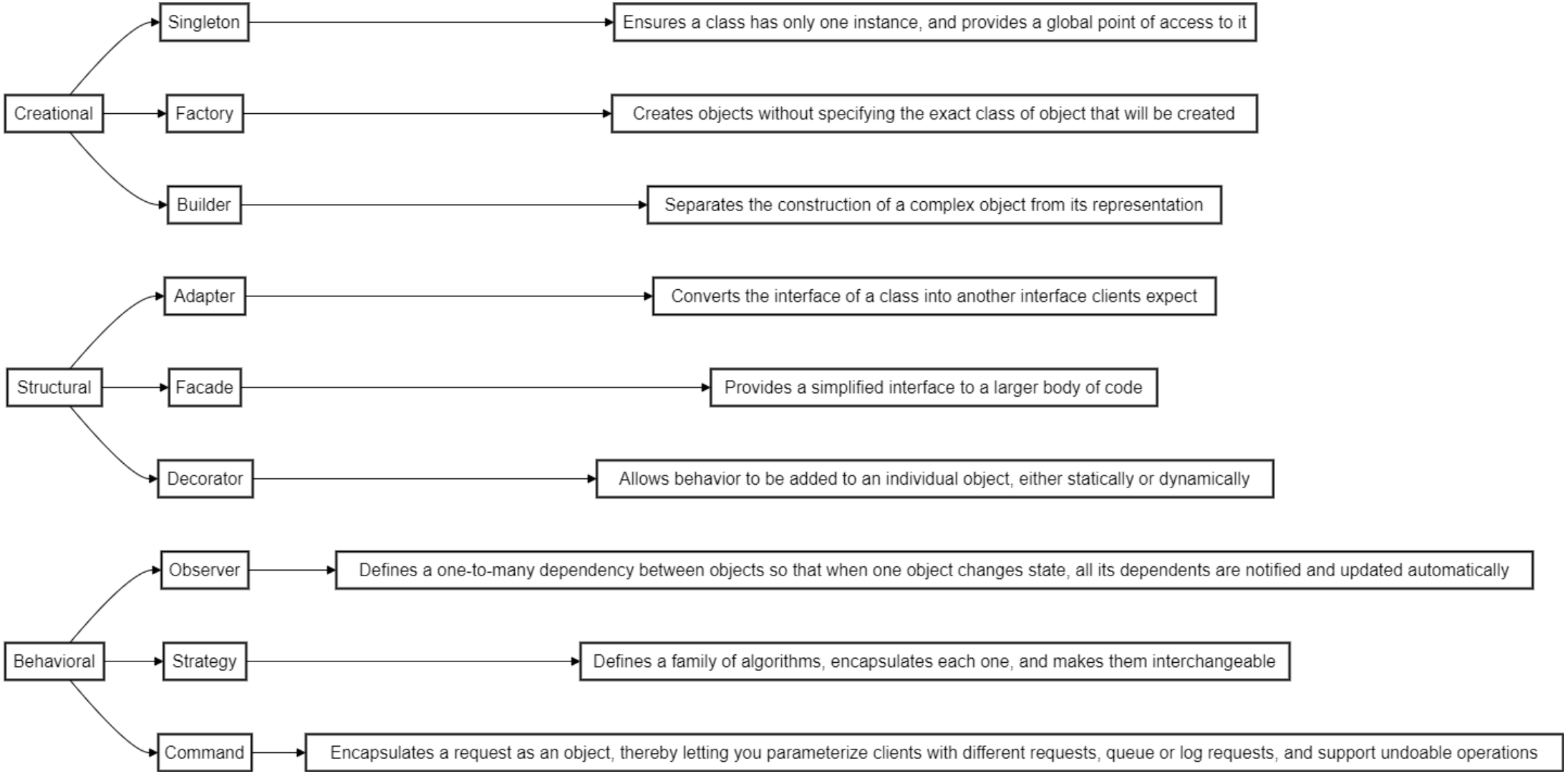
Design patterns

Design patterns are reusable solutions to common software design problems. They are often categorised into three main categories: creational, structural, and behavioural patterns.

- Creational patterns are concerned with object creation mechanisms, making it easier to create objects in a way that is flexible and compatible with the rest of the system. The Singleton Design Pattern is an example of a creational pattern. It ensures that only one instance of a class is created and provides global access to that instance. This is useful when a system needs to have a single instance of a class that can be accessed from multiple parts of the system. The Factory Method Pattern is another example of a creational pattern. It provides an interface for creating objects, but it allows subclasses to decide which class to instantiate. (Tutorialsteacher.com, 2023)

- Structural patterns are concerned with object composition and provide a way to create relationships between objects. Abstract Design Patterns are examples of structural patterns. They provide a flexible and extensible way to write code by separating the implementation details from the interface. Abstract Design Patterns allow developers to create abstractions or interfaces that can be implemented by multiple concrete classes. This makes it easy to change the implementation of a particular abstraction without affecting the rest of the system. The Adapter Pattern is another example of a structural pattern. It allows objects with incompatible interfaces to work together by creating a wrapper object that translates one interface to another. (Tutorialsteacher.com, 2023)
- Behavioural patterns are concerned with communication between objects and provide a way to manage interactions between different components. The Observer Pattern is an example of a behavioural pattern. It provides a way for objects to subscribe to events and be notified when those events occur. The Strategy Pattern is another example of a behavioural pattern. It defines a family of algorithms, encapsulates each one, and makes them interchangeable within a context. (Tutorialsteacher.com, 2023)

In summary, Abstract Design Patterns and Singleton Design Pattern are two different categories of design patterns. Abstract Design Patterns are examples of structural patterns that provide a flexible and extensible way to write code by separating the implementation details from the interface. Singleton Design Pattern is an example of a creational pattern that ensures that only one instance of a class is created and provides global access to that instance. The Factory Method Pattern is another example of a creational pattern, while the Adapter Pattern is another example of a structural pattern. The Observer Pattern and the Strategy Pattern are examples of behavioural patterns. All of these design patterns can be useful in different situations and can help developers write more flexible, extensible, and maintainable code. (Tutorialsteacher.com, 2023)


Creational
Singleton
Ensures a class has only one instance, and provides a global point of access to it
Factory
Creates objects without specifying the exact class of object that will be created
Builder
Separates the construction of a complex object from its representation
Structural
Adapter
Converts the interface of a class into another interface clients expect
Facade
Provides a simplified interface to a larger body of code
Decorator
Allows behavior to be added to an individual object, either statically or dynamically
Behavioral
Observer
Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
Strategy
Defines a family of algorithms, encapsulates each one, and makes them interchangeable
Command
Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations

Choosing the design pattern

Singleton Design Pattern:
Pros:

- Ensures that there is only one instance of a class created throughout the lifetime of the application, which can help to conserve resources and improve performance.
- Provides a centralised point of control for managing resources, which can help to simplify the design and implementation of the system.
- Can be easily extended to support additional functionality or features without impacting the rest of the system. (Tutorialsteacher.com, 2023)

Cons:

- Can be difficult to test, as the Singleton instance is often tightly coupled with other parts of the system, making it hard to isolate and test in isolation.
- Can introduce potential issues with concurrency and thread safety if not implemented correctly.
- Can make the code more difficult to understand and maintain, as the Singleton instance can be accessed from anywhere in the system, leading to potential side effects and dependencies. (Tutorialsteacher.com, 2023)

Abstract Design Pattern:
Pros:

- Provides a way to define a common interface or set of behaviours that can be shared across multiple classes or objects, promoting code reuse and modularity.
- Allows for flexibility and extensibility by allowing new classes to be easily added or modified without impacting the rest of the system.
- Can help to simplify the design and implementation of the system by abstracting away implementation details and focusing on high-level concepts. (Tutorialsteacher.com, 2023)

Cons:

- Can lead to code bloat and complexity if not used judiciously, as the number of abstract classes and interfaces can quickly grow and become difficult to manage.
- Can be difficult to understand and use correctly, especially for novice developers who may struggle with the concept of abstraction and how to apply it effectively.
- Can lead to performance issues if abstract classes and interfaces are not designed with performance in mind. (Tutorialsteacher.com, 2023)

Ultimately, the choice between Singleton and Abstract design patterns will depend on the specific needs and requirements of the Farm Central stock management website project. Both patterns have their strengths and weaknesses, and it is important to carefully consider the trade-offs involved when making a decision.
(Tutorialsteacher.com, 2023)

design patterns and architecture patterns chosen

Model-View-Controller (MVC) Pattern:
The Model-View-Controller (MVC) pattern is a popular architectural pattern used in web application development. It separates the concerns of an application into three distinct components: the model, the view, and the controller.
The model represents the data and the business logic of the application, the view represents the user interface, and the controller acts as an intermediary between the model and the view, handling user input and updating the model and view accordingly.
One of the main benefits of using the MVC pattern is that it promotes separation of concerns, making it easier to maintain and update the code. For example, changes to the user interface can be made without affecting the underlying data model, and changes to the data model can be made without affecting the user interface. This can also make it easier to test the different components of the application independently.
Singleton Design Pattern:(Tutorialsteacher.com, 2023) The Singleton design pattern is a creational pattern that is used to ensure that a class has only one instance, and that this instance can be accessed globally throughout the application.(Tutorialsteacher.com, 2023b)
One of the main benefits of using the Singleton pattern is that it can help to reduce memory usage and improve performance, as there is only one instance of the class that needs to be created and maintained throughout the application. This can be particularly useful in situations where multiple instances of a class could cause problems, such as in a database connection or a logging system.(Tutorialsteacher.com, 2023)
Another benefit of the Singleton pattern is that it can help to ensure consistency and prevent conflicts in situations where multiple objects need to access the same resource. For example, if multiple threads need to access a shared resource, using a Singleton can help to ensure that only one thread at a time is able to access the resource, preventing conflicts and improving performance. (Tutorialsteacher.com, 2023b)
Overall, the use of the MVC and Singleton design patterns can help to improve the structure, maintainability, and performance of your application, and can help to ensure that it is consistent and reliable.(Tutorialsteacher.com, 2023b)

the Accountant:

As a vital member of the Farm Central bid committee, we understand that you are concerned about the accuracy of the data and how it will affect the income of each farmer. That's why we propose to use the Model-View-Controller (MVC) design pattern for developing the stock management website.

The MVC design pattern is a well-known and widely used software design pattern that separates the application into three interconnected components: the model, view, and controller. The model represents the data and business logic, the view is the user interface, and the controller handles user input and updates the model and view accordingly.

By using the MVC design pattern, we can ensure that the data is organised and processed accurately and efficiently, which is essential for the success of the stock management website. The model component ensures that data is stored in a consistent format, making it easy to retrieve and process as needed. The view component is responsible for presenting the data to the user in a user-friendly and visually appealing manner, making it easy to understand and use. The controller component handles user input and updates the model and view components accordingly, ensuring that the data is always up-to-date and accurate.

Using the MVC design pattern also allows for easy testing and maintenance of the application. Since each component is separated and has its own distinct responsibilities, it is easy to test each component individually and ensure that it is functioning correctly. Additionally, since the components are loosely coupled, it is easy to modify and update the application as needed, without affecting other components of the application.

We believe that by using the MVC design pattern, we can develop a stock management website that meets and exceeds the expectations of Farm Central and its farmers. The accuracy and efficiency of the data processing, combined with the user-friendly and visually appealing user interface, will ensure that the stock management website is easy to use and understand, making it a valuable asset to the farmers who use it.

In summary, the use of the MVC design pattern for developing the stock management website will ensure that the data is organised and processed accurately and efficiently, while also allowing for easy testing and maintenance of the application. We are confident that this approach will result in a stock management website that meets and exceeds the expectations of Farm Central and its farmers.

The Head of Marketing:

As a visually oriented person, we understand that you care greatly about how quick and easy the process should be for the farmers that drop off stock. That's why we propose to use the Singleton design pattern for developing the stock management website.
The Singleton design pattern is a widely used software design pattern that ensures there is only one instance of a class throughout the application. This can improve performance and reduce memory usage, which is particularly important for applications with a lot of user interface elements. By using the Singleton pattern, we can ensure that the user interface is consistent and efficient, providing a seamless experience for the farmers using the stock management website.
The Singleton pattern also allows for easy integration with other systems and technologies, which can further enhance the functionality and usability of the application. This is particularly important for a stock management website, as it may need to integrate with other systems to ensure accuracy and efficiency of data processing.
Additionally, the Singleton pattern allows for easy customization and modification of the application, which can be important for addressing specific needs and requirements of Farm Central and its farmers. By using the Singleton pattern, we can ensure that the stock management website is visually appealing, user-friendly, and efficient, meeting and exceeding the expectations of Farm Central and its farmers.
In summary, the use of the Singleton design pattern for developing the stock management website will ensure that the user interface is consistent, efficient, and easy to use, while also allowing for easy integration with other systems and technologies. We are confident that this approach will result in a stock management website that is visually appealing, user-friendly, and efficient, meeting and exceeding the expectations of Farm Central and its farmers.

The Store Manager:

As an efficient person, we understand that you know the people working in the store will need to be trained to use the system. That's why we propose to use the Model-View-Controller (MVC) design pattern for developing the stock management website. The MVC design pattern is a well-known and widely used software design pattern that separates the application into three interconnected components: the model, view, and controller. This separation of concerns allows for easy training and understanding of the system, as each component can be learned and understood independently.

The model component represents the data and business logic of the application, while the view component is responsible for presenting the data to the user in a user-friendly and visually appealing manner. The controller component handles user input and updates the model and view components accordingly, ensuring that the data is always up-to-date and accurate.

By using the MVC design pattern, we can ensure that the stock management website is efficient, user-friendly, and easy to learn and understand. The separation of concerns provided by the MVC pattern makes it easy to train users on specific components of the application, without requiring them to understand the system as a whole. This can be particularly important for employees who may have varying levels of technical expertise.

Additionally, the MVC pattern allows for easy customization and modification of the application, which can be important for addressing specific needs and requirements of the store and its employees. By modifying the model or controller components, for example, we can tailor the application to meet specific needs or requirements, without affecting other components of the application.

In summary, the use of the MVC design pattern for developing the stock management website will ensure that the system is easy to learn and use, while also allowing for easy customization and modification to meet specific needs and requirements. We are confident that this approach will result in a stock management website that is efficient, user-friendly, and easy to learn and understand, meeting and exceeding the expectations of Farm Central and its employees.

Conclusion

In conclusion, the non-functional requirements of security, performance, and usability are of high importance in the development of the stock management website for Farm Central. We plan to address these requirements by implementing industry-standard security measures, optimising the website's code and database design, and conducting user testing. Additionally, design patterns such as the MVC and Repository patterns, as well as architecture patterns such as the SOA pattern, are relevant to the project and will be used to develop a scalable, maintainable, and modular system.

Referencing

Requirements in Software Engineering (International Series in Software Engineering). Springer US.
Glinz, M., 2007. On Non-Functional Requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007). pp. 21-26.
Lawrence, K.D. and Barker, K., 2001. Integrating non-functional requirements into UML models. In: Proceedings First International Workshop on Scenario-Based Requirements Analysis (SITRE '01). IEEE, pp. 1-6.
Mylopoulos, J., Chung, L. and Nixon, B., 1992. Representing and using nonfunctional requirements: A process-oriented approach. IEEE Transactions on Software Engineering, 18(6), pp.483-497.
Robertson, S. and Robertson, J., 2006. Mastering the Requirements Process (2nd ed.). Addison-Wesley Professional.
Martin, R.C., 2002. Agile Software Development: Principles, Patterns, and Practices. Prentice Hall.
Martin, R.C., 2003. UML for Java programmers. Cambridge University Press.
Martin, R.C., 2008. Clean code: a handbook of agile software craftsmanship. Pearson Education.
Martin, R.C., 2014. Clean architecture: a craftsman's guide to software structure and design. Prentice Hall.
Smith, M.J. and Biddle, R., 2002. UML 2.0 Pocket Reference. O'Reilly Media, Inc.
Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software (1st ed.). Addison-Wesley.
Freeman, E., & Freeman, E. (2004). Head first design patterns: a brain-friendly guide. O'Reilly.
Shalloway, A., & Trott, J. R. (2004). Design patterns explained: a new perspective on object-oriented design (2nd ed.). Addison-Wesley.
Ambler, S. W. (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. John Wiley & Sons.
Horswill, I. (2007). Design patterns in C#. O'Reilly.
AltexSoft. (2019). *Non-functional Requirements: Examples, Types, How to Approach*. [online] Available at: https://www.altexsoft.com/blog/non-functional-requirements/#:~:text=Non%2Dfunctional%20requirements%20or%20NFRs,reliability%2C%20data%20integrity%2C%20etc. [Accessed 14 Apr. 2023].

Pentalog. (2022). *SOLID Principles in Object Oriented Programming*. [online] Available at: https://www.pentalog.com/blog/it-development-technology/solid-principles-object-oriented-programming/ [Accessed 14 Apr. 2023].

Tutorialsteacher.com. (2023a). *ASP.NET MVC Tutorials*. [online] Available at: https://www.tutorialsteacher.com/mvc [Accessed 14 Apr. 2023].

Tutorialsteacher.com. (2023b). Available at: https://www.tutorialsteacher.com/csharp/design-patterns [Accessed 14 Apr. 2023].