**Ediz Alperen Amin Yurdakul  
ST10242585  
PROG7311 POE PART 1**

**Feedback:** Not 3rd year standard of work. Page numbers, referencing…how will the concepts be implemented? **Changelog**:

* Added page numbers
* Added table of contents
* Added new reference
* Added a new section implementation of the concepts.

Table Of Contents

[**Introduction 1**](#_b5doxm6bahe5)

[**Non-Functional Requirements 2**](#_uepnu6xh0u8q)

[**Design Patterns and Architecture Patterns 3**](#_rh8nxp3t583t)

[**Implementation of Non-Functional Requirements and Design Patterns 4**](#_uog92ytsuy9r)

[**Conclusion 5**](#_6nlc2h802nui)

[**References 6**](#_32dnbclcmqwp)

# 

# Introduction

The primary aim of this report is to discern non-functional requirements that are of the utmost importance, and to deliberate on the pivotal role played by software design patterns in addressing these requirements.

As elucidated by Sommerville (2007), non-functional requirements are not explicitly linked to the functions or operations that the system performs. Rather, they embody a diverse range of attributes such as performance, usability, security, maintainability, and reliability. Moreover, Sommerville (2007) expounds on how these stipulations delineate the degree of competence required from the software in carrying out specific functions, rather than merely enumerating what those functions may be.

Design patterns are repeatable solutions to common problems that arise during software design and development. They provide a standardised approach to solving these problems and are specific to certain scenarios. By using design patterns, developers can take advantage of the best solutions to specific problems faced during software development. Design patterns can improve software quality by promoting code reuse, standardisation, and maintainability.

# 

# Non-Functional Requirements

Developing a successful web application for Farm Central requires thorough attention to non-functional requirements. These requirements express how well the system should perform specific tasks rather than what the tasks are per se. Essential non-functional requirements include fast and efficient performance, user-friendly usability, robust security, easy maintenance, and scalable infrastructure. By placing these necessities at the forefront of our priorities, we can ensure that the Farm Central web application meets the high standards necessary for success.

* **Performance:** The web application needs to execute speedily and efficiently with minimal disruptions to data entry and exit to guarantee an uninterrupted buttery smooth user experience.
* **Usability:** The web application's interface should be user-friendly, enabling farmers to learn and use the website promptly and to deliver their stock.
* **Security:** The security of farmers' personal information, especially payment information, is critical. Implementing the necessary measures to safeguard the web application against unauthorised access and ensure its security is crucial.
* **Maintenance:** A web application is a complex system requiring regular upkeep and maintenance. Consequently, it is essential to make updating and maintenance facile and swift to keep the application running smoothly.
* **Scalability:** The Farm Central web application must be scalable to meet future exigencies. Solanki (2018) explains scalability entails ensuring the application runs proficiently even when the workload increases, such as more farmers join Farm Central. It is imperative to consider the application's ability to handle augmented traffic and user load and add new features and functionality when necessary.

By addressing these non-functional requirements, we can ensure that the Farm Central web application meets the necessary performance, usability, security, maintenance, and scalability standards required for success.

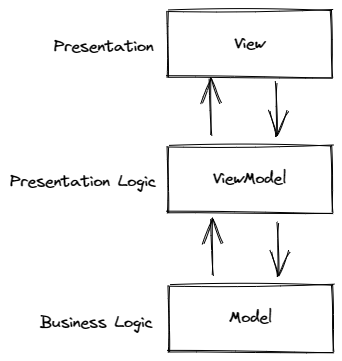
# 

# Design Patterns and Architecture Patterns

Design and architecture patterns are relevant to the Farm Central web application project. Riehle and Züllighoven (1996) explains patterns are essential when addressing the common problems that invariably crop up during software design and development. By implementing best practices, these patterns enhance the quality of the software, reduce development time, and promote much-needed code reuse.

Design patterns come with a host of applications that can be leveraged to improve software design and development. They help developers identify common problems, provide workable solutions for those problems, and allow for easy modification and maintenance of the system. Additionally, architectural patterns establish a sturdy foundation for the system's overall structure, guiding the development of individual components with unparalleled ease.

Some specific design patterns that could prove helpful in the Farm Central web application include the Model-View-ViewModel (MVVM) pattern, the Facade pattern, and the Strategy pattern. The MVVM pattern allows separations of concern of an application's business and presentation logic from its user interface (UI). This separation between application logic and the UI can effectively tackle numerous development issues. The Facade pattern significantly simplifies interactions between subsystems by providing a single interface, while the Strategy pattern allows for different algorithms or behaviours to be selected at runtime, offering unparalleled flexibility

.

# 

# Implementation of Non-Functional Requirements and Design Patterns

When developing the Farm Central web application, we can utilise specific design patterns to tackle common design challenges and improve the overall quality of our software.

One such pattern is the Model-View-ViewModel (MVVM), which helps us separate the business logic and data from the user interface (UI). Implementing MVVM ensures our code is more organised, maintainable, and testable. In the Farm Central web application, we can have a model representing the data and business logic, such as farmer profiles, stock information, or transaction details. The view renders the UI elements, allowing farmers to interact with the application. The ViewModel acts as a middleman between the model and the view, handling the presentation logic and providing data and commands to the view. For example, a ViewModel for managing stock can retrieve data from the model, perform calculations, and update the view accordingly.  
  
The Facade pattern simplifies interactions between complex subsystems. Our web application may have various subsystems responsible for tasks like inventory tracking or payment processing. By creating a facade, we provide a simplified and unified interface that hides the complexities of these subsystems, making it easier for other components to interact with them without needing to understand their internal workings. For instance, we can create a StockManagementFacade that exposes methods like addStock(), updateStock(), or getStockDetails(), internally coordinating the interactions between subsystems.

By incorporating these design patterns into our software development process, we ensure that our code is well-structured, maintainable, and adaptable to future changes. The MVVM pattern separates concerns, managing the business logic and UI. The Facade pattern simplifies subsystem interactions, promoting a more intuitive and streamlined architecture. The Strategy pattern allows us to dynamically switch between different algorithms or behaviours, offering flexibility and extensibility.

Implementing these design patterns in the Farm Central web application helps us create a high-quality software product that meets the needs of farmers efficiently and effectively.

# 

# Conclusion

Non-functional requirements are undoubtedly a necessary aspect of the software development process. These requirements outline how well a system ought to perform specific tasks. Prioritising non-functional requirements, such as performance, usability, security, maintenance, and scalability, is crucial for the Farm Central web application to meet the high standards necessary for success. The judicious use of design and architecture patterns can offer optimal solutions to common problems in software development and drive code reuse, standardisation, and maintainability. Several specific design patterns, including MVVM, Facade, and Strategy patterns, could prove promising in boosting the quality of the Farm Central web application.

Strongly recommend that the bid committee prioritise non-functional requirements and weigh on deploying design and architecture patterns to elevate software development. Pursuing these best practices can help the committee develop a high-quality software product that caters to Farm Central's and its users' needs with unprecedented finesse, thereby ensuring unbridled triumph.

# 

# References

* Sommerville, I. 2007. *Software engineering*. 8th ed, Harlow, Essex: Addison-Wesley.
* Riehle, D. and Züllighoven, H. 1996. Understanding and using patterns in software development. *Tapos*, *2*(1): 3-13.
* Solanki, J. 2018. How to Build a Scalable Application up to 1 Million Users on AWS, Insights on Latest Technologies - Simform Blog, December 10, 2018. [Blog]. Available at: <https://www.simform.com/blog/building-scalable-application-aws-platform> [Accessed 21 June 2023].