PROG7311

Atlegang Mogatusi st10357922

Part 1

The Agri-Energy Connect platform plans on using green energy to help better South African agriculture through sustainable farming. They will also build a digital ecosystem that connects farmers, green energy experts, and enthusiasts who will be able to collaborate, share resources, and innovate in the realms of sustainable agriculture and renewable energy.

Non-functional requirements analysis

To ensure that the platform meets user expectations the following non-functional requirements need to be included such as security, scalability, usability, performance, and maintainability. (Saleh, A. M., & Alshayeb, M. 2021)

* Security: The user’s transactions, information, farming techniques, water conservation methods and everything that is shared on the platform, this data needs to be stored and protected using strong security measures. For data privacy employ POPIA principles. You can also test the platform regularly for any vulnerabilities, employ SSL encryption, and secure APIs. (Kvartalnyi, 2023; Krüger, 2024; Inoxoft, 2023).
* Usability: The user interface design for the platform needs to be easy to use so that the platform’s users (farmers, green energy experts, and enthusiasts) are able to easily navigate the platform. Ensure that the platform meets Web Content Accessibility Guidelines (WCAG). (Krüger, 2024; Inoxoft, 2023)
* Scalability: The system should be able to grow to accommodate many users at once. Cloud services like azure can be used to scale automatically and to load balance. (Krüger, 2024; Inoxoft, 2023)
* Performance: The system should be able to load swiftly, process large volumes of user interactions, and also handle real-time data sharing. (Krüger, 2024; Inoxoft, 2023) Maintainability: A solution's or its component's maintainability is the amount of time required to be fixed, modified to improve performance or other attributes, or adjusted to a changing environment. (Krüger, 2024; Inoxoft, 2023)

The impact of these non-functional requirements on my agile development approach is that it emphasises developing functional software rapidly, dealing with clients regularly, and being flexible enough to adjust to changes with ease. (Perforce, 2024)

Role of design and architecture patterns

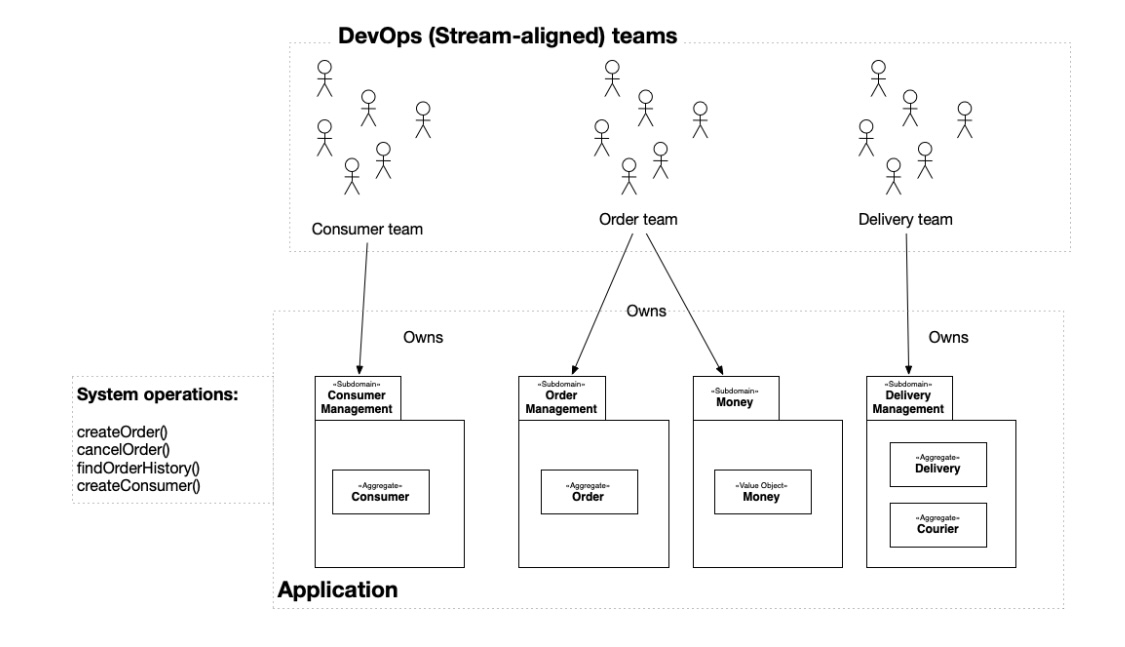
The role of design patterns and architecture patterns are relevant in the context of this project because design and architecture that are good can improve the software system’s performance. It also ensures that the processes are efficient and quick in a well designed software system with good design and architecture patterns. Scalability is a major advantage of employing good architecture and design patterns. For businesses that are expanding, scalable systems are crucial because they can effortlessly manage growing numbers of users and transactions. Future updates and enhancements may be less expensive if the system is modular and simple to maintain, which is another benefit of good architecture and design patterns. Businesses can boost their overall efficiency and enhance the functionality of their software systems by investing in sound architecture and design patterns. (Horta, 2023)

Design pattern suggestions:

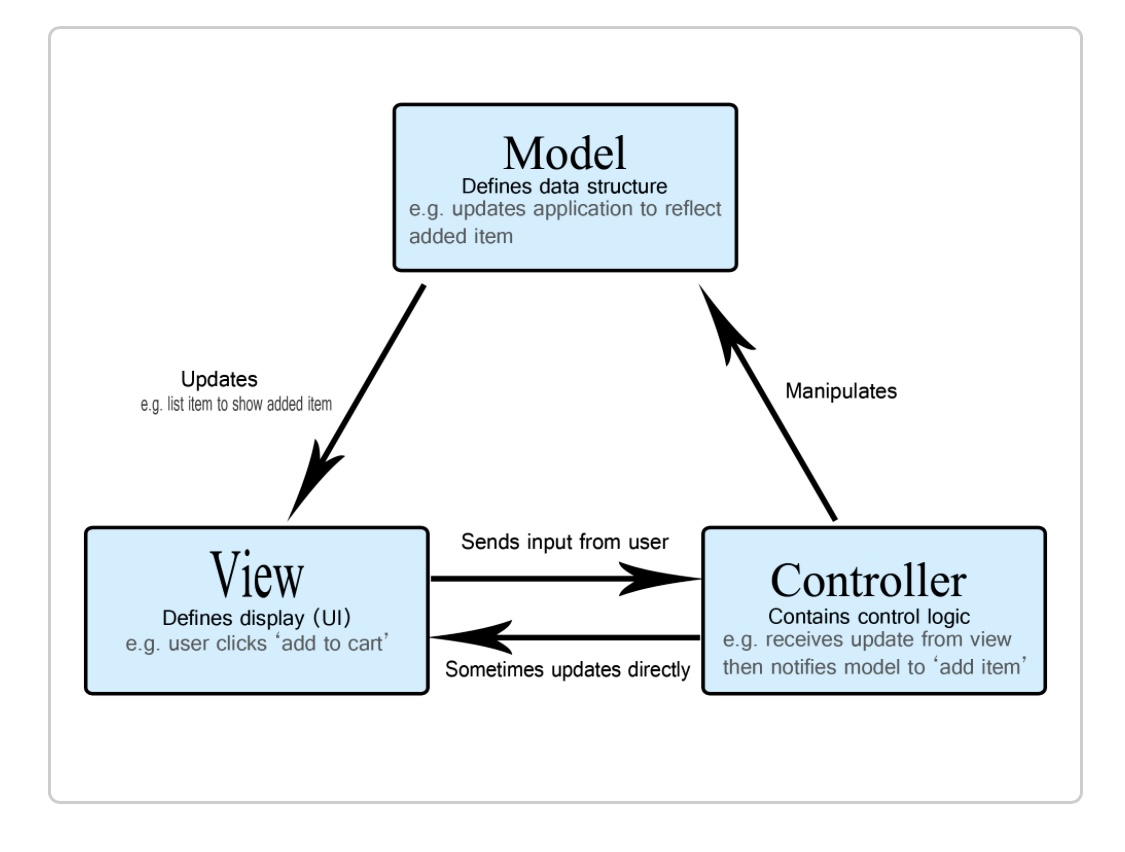
* Factory method pattern: The Factory Method is a creational design pattern that lets subclasses modify the kind of objects that are created while still offering an interface for producing objects in a superclass. The different user roles like the farmer, green energy expert and enthusiast will be managed by this method. (Factory Method, 2025)
* Observer pattern method: A subscription mechanism can be implemented using the Observer behavioural design pattern to alert numerous objects to any changes made to the object they are viewing. (Observer, n.d.)

Architecture pattern suggestions:

* Microservices pattern: An alternate method for creating apps is the microservices architecture. This method builds a larger application from several modular services that interact with one another using APIs. Together with a particular business objective, these loosely connected, independently deployable services make use of separate databases and coding. A Microservices architecture usually assigns responsibility for each service to a separate team. This makes it possible to scale, deploy, test, and update each service independently while maintaining reasonable complexity. (Microservice Architecture pattern, 2025; Davis, 2023)

The image below was taken from: Microservice Architecture pattern. (n.d.). Retrieved from microservices: https://microservices.io/patterns/microservices.html

* Model-view-controller (MVC) pattern: According to the model-view-controller (MVC) pattern, an application is composed of three parts. The view shows data and communicates with the user, the controller manages user input and acts as a bridge between the view and the model, and the model houses the application's data and primary functionality. Although this approach allows the program to produce multiple views, the complexity is increased by the layers of abstraction. (MVC, n.d.)

The image below was taken from: MVC. (n.d.). Retrieved from mdn web docs: https://developer.mozilla.org/en-US/docs/Glossary/MVC

Efficiency, modularity, and reusability are promoted by the patterns above. Testability and code maintenance is enhanced by MVC, Scalability is supported by microservices. In collaborative settings, dynamic and responsive user experiences are made possible by the application of observer patterns.

An innovative digital initiative built on sustainability and innovation is the Agri-Energy Connect platform. For South Africa’s agricultural and green energy markets, this report offers a reliable, secure, and scalable solution by combining strong non-functional requirements with good design and architecture patterns.

# References

Davis, A. (2023, December 28). *An In-Depth Guide to Microservices Design Patterns*. Retrieved from OpenLegacy: https://www.openlegacy.com/blog/microservices-architecture-patterns/

Davis, A. (2023, October 9). *Monolithic vs Microservices Architecture: Pros, Cons and Which to Choose*. Retrieved from OpenLegacy: https://www.openlegacy.com/blog/monolithic-application

*Factory Method*. (n.d.). Retrieved from Refactoring guru: https://refactoring.guru/design-patterns/factory-method

Horta, J. Y. (2023, March 27). *The Importance of Good Architecture and Design Patterns for High-Performance Software*. Retrieved from linkedin: https://www.linkedin.com/pulse/importance-good-architecture-design-patterns-software-jos%C3%A9-yadiel

Krüger, G. (2024, August 8). *Non-functional Requirements: What They Do, Examples, and Best Practices*. Retrieved from Perforce: https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples

Kumar, P. (2023, December 17). *Understanding and Implementing Design Patterns in Your Projects*. Retrieved from Medium : https://codewithpawan.medium.com/understanding-and-implementing-design-patterns-in-your-projects-467bc53f9690

Kvartalnyi, N. (2023, December 6). *Functional Vs. Non-Functional Requirements: Why Are Both Important?* Retrieved from inoxoft: https://inoxoft.com/blog/functional-vs-non-functional-requirements-why-are-both-important/#:~:text=Similarly%2C%20neglecting%20non%2Dfunctional%20requirements,unresponsive%2C%20or%20prone%20to%20crashes

*Microservice Architecture pattern*. (n.d.). Retrieved from microservices: https://microservices.io/patterns/microservices.html

*MVC*. (n.d.). Retrieved from mdn web docs: https://developer.mozilla.org/en-US/docs/Glossary/MVC

n/a. (n.d.). *What is Agile Development?* Retrieved from opentext: https://www.opentext.com/what-is/agile-development

*Observer* . (n.d.). Retrieved from Refactoring guru: https://refactoring.guru/design-patterns/observer

Walker, V. (2022, March 16). *14 software architecture design patterns to know*. Retrieved from Red Hat Blog: https://www.redhat.com/en/blog/14-software-architecture-patterns

Gross, F., & Yu, E. (2001). “From Non-Functional Requirements to Design through Patterns. Retrieved from: https://link.springer.com/article/10.1007/s007660170013

Ameller, D., Franch, X., & Cabot, J. (2019). Software Architectural Patterns in Practice: An Empirical Study. Empirical Software Engineering, 24(1), 320–364. Retrieved from: https://link.springer.com/article/10.1007/s10664-018-9620-7

de la Vara, J. L., & Sánchez, J. (2010). Capturing Software Architecture Knowledge for Pattern-Driven Design. Journal of Systems and Software, 83(12), 2322–2337. Retrieved from: https://www.sciencedirect.com/science/article/abs/pii/S0164121210001676

Saleh, A. M., & Alshayeb, M. (2021). A Systematic Review of Non-Functional Requirements Mapping into Architecture Styles. Bulletin of Electrical Engineering and Informatics, 10(4), 1957–1969. Retrieved from: https://www.beei.org/index.php/EEI/article/view/4081

Kruchten, P. (1995). Architectural Blueprints—The ‘4+1’ View Model of Software Architecture. IEEE Software, 12(6), 42–50. Retrieved from: https://ieeexplore.ieee.org/document/4766589