PROG7311

PART 1

PORTFOLIO OF EVIDENCE

Agri-Energy Connect Platform

ST10282051

8/04/2025

Table of Contents

Date: 18/04/2025

Agri-Energy Connect Platform:

A Sustainable Digital Solution for Thaba Verde Farm

The Agri-Energy Connect Platform is a digital ecosystem designed to integrate sustainable agriculture and renewable energy solutions. This platform will provide a centralized hub for collaboration, knowledge-sharing, and resource access among farmers, energy experts, and sustainability advocates. The platform aims to help Thaba Verde Farm and other agricultural stakeholders adopt eco-friendly, cost-effective, and energy efficient farming practices.

Objectives

- **Develop a Scalable Platform** – Create a cloud-based system that adapts to increasing users and ensures high performance. For example, as more farmers join the platform, the system will scale seamlessly to handle more users without slowing down.
- **Provide Sustainable Energy Solutions** – Offer an interactive marketplace where farmers can explore solar, wind, and biogas technologies. For example, farmers can browse solar panel solutions suitable for powering irrigation systems.
- **Enhance Knowledge and Collaboration** – Implement training modules and discussion forums to educate users on sustainable farming. For instance, webinars on water conservation techniques can be hosted.
- **Ensure Security and Data Integrity** – Use robust security measures to protect user and financial data. Sensitive farm data, like financial transactions or farm practices, will be encrypted and securely stored.
- **Enable Funding Opportunities** – Connect farmers with government grants and investment partners to support green agriculture projects, like applying for subsidies to install renewable energy systems (John W. Satzinger, 2015).

Purpose

The Agri-Energy Connect project will help Thaba Verde Farm, a commercial livestock farm in South Africa, transition to sustainable agricultural practices and renewable energy solutions. The platform will address challenges like high energy costs and water scarcity by offering resources and technologies such as solar energy for water pumping and biogas for energy needs. Built using Visual Studio C# (MVC framework) and SQL Server Management Studio (SSMS) for secure data management, the platform will ensure that sensitive data is protected (Sommerville, 2016).

Agri-Energy Connect aims to

- Provide sustainable farming resources to help improve livestock management. For example, access to guides on pasture management and eco-friendly feed options.
- Offer a marketplace for affordable green energy solutions tailored to agriculture, such as solar-powered irrigation systems.

- ✞ Deliver educational materials to upskill farm workers in renewable energy, like training modules on installing and maintaining solar panels.
- ✞ Enable collaborative projects and funding opportunities to support sustainability, such as a forum for farmers to share successful green energy adoption experiences.
- ✞ Ensure real-time data sharing and secure decision-making, like providing weather forecasts and energy usage reports to optimize farming operations (Sommerville, 2016).

Scope of the Agri-Energy Connect Project

The Agri-Energy Connect platform will support Thaba Verde Farm in adopting sustainable practices through collaboration with green energy providers and sustainability professionals. By using Visual Studio C# (MVC framework) and SQL Server Management Studio (SSMS) for development, the platform will address key issues like high operational costs and limited access to renewable technologies. It will also provide farmers with tools and resources to drive innovation in Agri-tech, promoting long-term sustainability (Sommerville, 2016).

System Functionalities	
For Farmers	• Add new products with: ○ Name, category, production date, and image • View and manage their own products

For Employees	• Add farmer profiles with personal details
	• View products from any farmer
	• Filter products by:
	  ○ Date range
	  ○ Category/type

For All Users(Farmers and Employees)	• View and post to the news feed • Comment on community posts • Community can view post but not comment
User Roles	• Role Description Farmer Can add/view products, and participate in news feed Employee Can manage farmer profiles, view all farmer products, and filter by criteria Admin • Has full access to dashboards and user management Guest Can view public pages only

System Requirements Specifications

Requirements analysis is a crucial step in ensuring the success of a system or software platform. Requirements are typically categorized into two types: Functional and Nonfunctional. Functional requirements describe the specific actions or features the platform must perform, such as allowing users to browse the green energy marketplace or participate in discussion forums. On the other hand, Non-functional requirements outline the platform's overall performance and qualities, including aspects like security, scalability, usability, and speed, ensuring that the system functions effectively and meets user expectations (GeeksForGeeks, 2025).

Analysis of Non-Functional Requirements

What are the non-functional requirements for the platform?

In the case of Agri-Energy Connect, the non-functional requirements focus on ensuring that the platform not only meets the functional needs of sustainable farming and green energy but also performs efficiently, securely, and reliably. These requirements are essential for providing a seamless and satisfying user experience for farmers, energy experts, and sustainability advocates who will rely on the platform for their day-to-day operations and collaborative efforts in advancing green energy in agriculture (Grow Solutions, 2023).

Parameters	Functional Requirement	Non-Functional Requirements
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case	It is captured as a quality attribute
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software	Helps you to verify the performance of the software
Area of focus	Focuses on user requirement	Concentrates on the user`s expectation and experience
Documentation	Describe what the product does	Describes how the product works
Product Info	Product features	Product properties

Jelvix jelvix.com

Figure 1: Difference between Functional and Non-Functional Requirements (Llyukha, n.d.).

Non-functional requirements for the Agri-Energy Connect System

URPS/S

Usability	The platform will be user-friendly and accessible to a diverse range of users, including farmers with varying levels of technical expertise. It will have an intuitive design, with easy navigation across all features (e.g. marketplace, educational resources, project collaboration). For example, the farm manager at Thaba Verde Farm, with minimal tech experience, will be able to easily navigate through the Green Energy Marketplace to compare solar-powered irrigation systems without requiring advanced technical knowledge. The platform will present clear, simple instructions and have a streamlined process for selecting and purchasing energy-efficient solutions.
Reliability	The platform will ensure high availability, with minimal downtime. In case of failure, it will recover quickly, ensuring continuous access to all features, such as resource sharing, forums, and green energy solutions. The system will operate without interruptions, providing consistent service. For example, If Thaba Verde Farm is conducting a live training webinar on sustainable farming practices, the platform will remain stable and accessible throughout the event. If any technical issues arise, such as server downtime, the platform will recover swiftly to prevent any disruption in the farm's ongoing training or collaboration.
Performance	The platform will deliver fast response times, minimizing latency when users interact with the system. Actions like searching for products in the marketplace, accessing forums, or enrolling in training should happen in a timely manner to improve the overall user experience. For example, when Thaba Verde is searching for a biogas energy solution in the marketplace, the results will load quickly, allowing the farm manager to immediately compare product features and make an informed decision. This ensures that the time spent on decision-making is minimized, and the farm can quickly proceed with the adoption of green energy solutions.

Scalability	The platform must be able to scale to accommodate a growing number of users and interactions. As more farmers and green energy providers join the platform, it should handle increasing traffic, transactions, and data without performance degradation. For example, as Thaba Verde continues to adopt new sustainable energy systems, other farms in the region join the platform. The platform must be able to handle a surge in users during peak times, such as when there are live collaboration projects or community discussions about implementing solar irrigation systems, without slowing down or crashing.
Security	The platform must implement robust security measures to protect sensitive user data, including financial transactions and proprietary farming techniques. End-to-end encryption, secure authentication, and regular security updates should be in place to safeguard users' privacy and data integrity. For example, Thaba Verde Farm might upload sensitive information, such as financial details for grant applications or proprietary farming techniques, onto the platform. The platform must encrypt all such data and ensure that only authorized personnel have access to it. Additionally, any financial transactions related to purchasing energyefficient systems must be securely processed with multi-factor authentication to prevent unauthorized access.

(Adams, 2015).

Implementation of Non-Functional Requirements for the System

NFR	Implementation	Impact
Usability	User-focused MVC views with clean layouts and icons tailored for farmers and sustainability partners. For example, the solar funding application form guides farmers step-by-step with simple language and tooltips (Information Services, n.d.).	This requirement will prioritize UI/UX design and user feedback loops. We will have dedicated phases for designing and testing the interface. Development will focus on keeping the platform simple, intuitive, and accessible, meaning that complex features will need to be presented in a user-friendly manner. Sprint Planning: More time will be allocated in the initial phases for UI/UX design and testing, with frequent revisions based on user feedback (Information Services, n.d.).
Reliability	Built-in error handling in C#, regular SSMS backups, and logging with alerts for crashes. failures (Adams, 2015). For example, If the energy usage tracking tool fails, the system logs the issue and notifies support, preventing data loss (Adams, 2015).	Maintains uptime and ensures quick recovery from system
Performance	Optimized SQL queries and caching frequent data like course lists. slower internet (IPH Technologies, 2024). For example, the “Marketplace” page loads instantly by caching product categories and pricing (IPH Technologies, 2024).	Speeds up user experience, especially in rural areas with training
Scalability	Modular MVC structure and indexed SSMS tables to Supports in users and data. performance (GeekForGeeks, 2024)..	platform growth without sacrificing speed or manage growth

For example, as more farmers join, user data is indexed by province to keep dashboard loading times low (GeekForGeeks, 2024).

9

Security	HTTPS, role-based access in C# controllers, and AESencrypted sensitive data in SSMS. - Access Control – Implementing role-based access control (RBAC), ensuring that only authorized users (e.g. farm managers, energy providers) can access specific data or features. For example, only verified investors can access project financials, while farmers only see their own data (Ciampa, 2021).	Security will influence the design of user authentication and data storage systems, as well as encryption practices. The security implementation will require a dedicated effort in planning for user data protection, compliance with privacy regulations (e.g. GDPR, POPIA), and testing for vulnerabilities. Development will require ongoing attention to security patches and updates to ensure the platform is resilient to new threats (Ciampa, 2021).

10

Role of Design and Architecture Patterns

Design and Architecture Patterns in the AgriEnergy Connect Platform

What are Design Patterns - Design patterns are proven solutions to common software design challenges. They help developers write structured, maintainable, and scalable code by following best practices. Rather than solving the same problem repeatedly, developers use these patterns as blueprints to build efficient systems (GeeksForGeeks, 2025). **Key Benefits of Design Patterns**

- Reusability – Common solutions can be applied across multiple projects, saving time and effort.
- Standardization – Establishes a shared structure and best practices among developers, improving collaboration.
- Efficiency – Reduces development time by avoiding redundant problem-solving.
- Flexibility – Allows systems to be easily modified and expanded without breaking functionality.

For the Agri-Energy Connect platform, choosing the right design patterns will ensure scalability, flexibility, and maintainability while addressing specific challenges in user management, system integration, and performance optimization (GeeksForGeeks, 2025).

Relevance of Design Patterns

Creational Patterns – Ensuring Efficient Object Creation	
Factory Method & Abstract Factory	The platform may need to generate different types of users (farmers, researchers, green energy providers). Using a Factory Method Pattern, the system can dynamically create the right user type without modifying core code. For example, A *UserFactory* class could instantiate different user roles (Farmer, EnergyExpert, Researcher) based on user registration details.
Singleton Pattern	Certain platform-wide components, like a centralized database connection or a configuration manager, should only have one instance running at a time. A Singleton ensures that these components remain consistent and globally accessible. For example, A *DatabaseConnection* class ensures a single database instance for managing user profiles, energy resources, and transactions.

Builder Pattern	The platform might need to generate customized farm reports (energy consumption, crop yield, sustainability impact). The Builder Pattern allows step-by-step creation of complex reports with different components.
	For example, A *ReportBuilder* could generate customized sustainability reports, selecting specific metrics like water usage, solar power efficiency, and carbon footprint.

(Kumar, 2024).

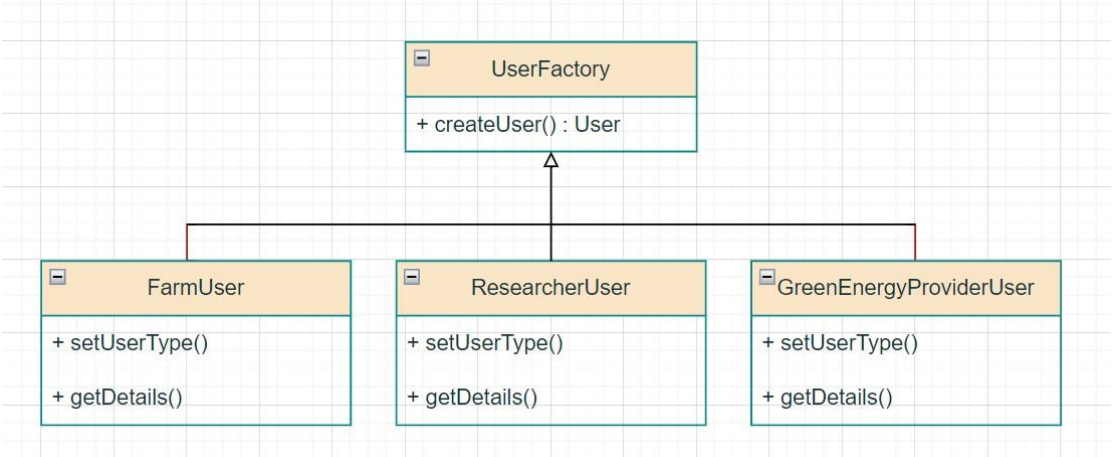

Figure 2: Factory Method flowchart (GeeksForGeeks, 2024)**.**

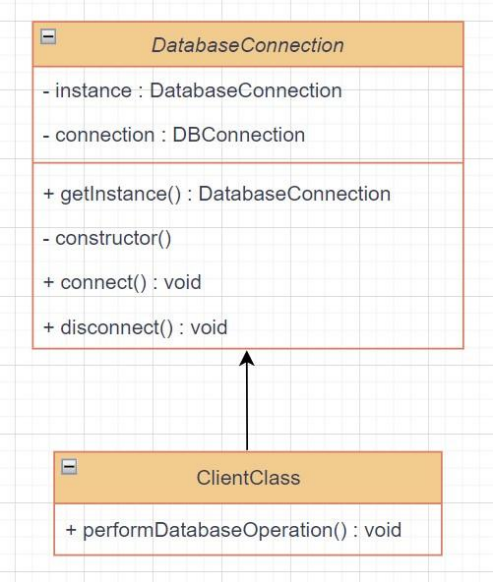

Figure 3: Implementation of Singleton Pattern (GeeksForGeeks, 2025)**.**

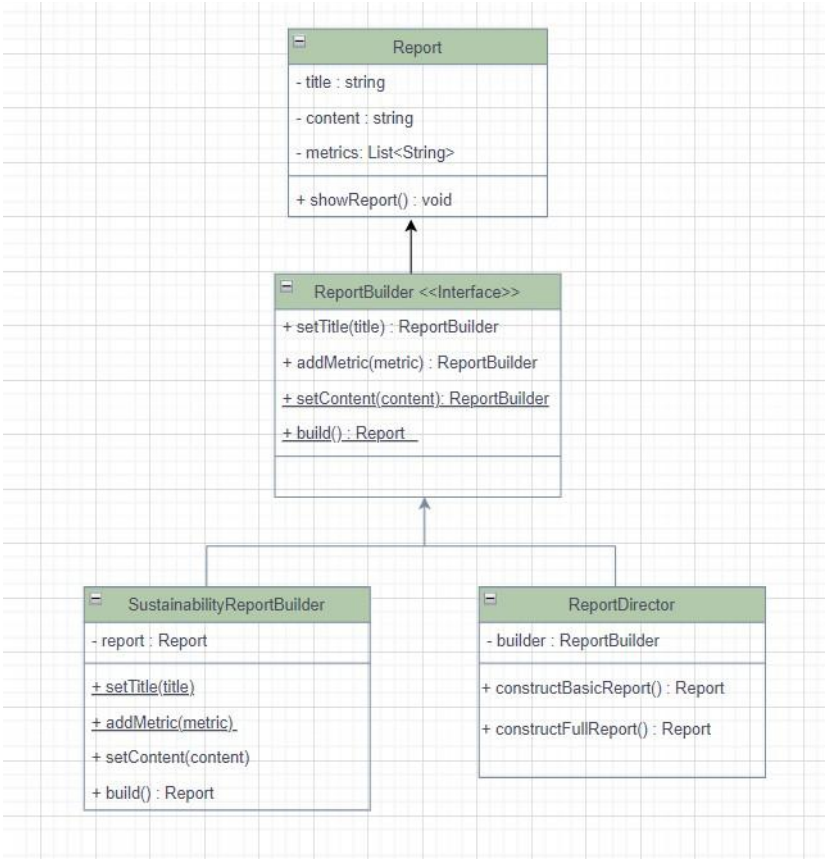

Figure 4: Builder Pattern (GeeksForGeeks, 2025)**.**

Structural Patterns – Organizing System Components Efficiently	
Adapter Pattern	The platform may need to integrate with third-party energy monitoring tools, government databases, or IoT devices. An Adapter Pattern allows seamless data exchange between systems with different interfaces. For example, an EnergyDataAdapter could convert data from different solar energy providers into a standard format for the AgriEnergy Connect dashboard.
Facade Pattern	The platform could have complex backend logic for calculating farm energy needs, subsidy eligibility, and crop sustainability ratings. Instead of exposing this complexity, a Facade Pattern provides a simplified interface. For example, a SustainabilityFacade class could offer a single API endpoint for retrieving a farm's renewable energy score, even if it pulls data from multiple sources.
Composite Pattern	The platform might deal with hierarchical data structures, such as farms grouped by region, or energy grids with multiple solar panels. A Composite Pattern allows treating both individual farms and farm groups uniformly.

For example, a *FarmCluster* object could represent a single farm or a group of farms, allowing scalability in resource distribution planning.

(Kumar, 2024).

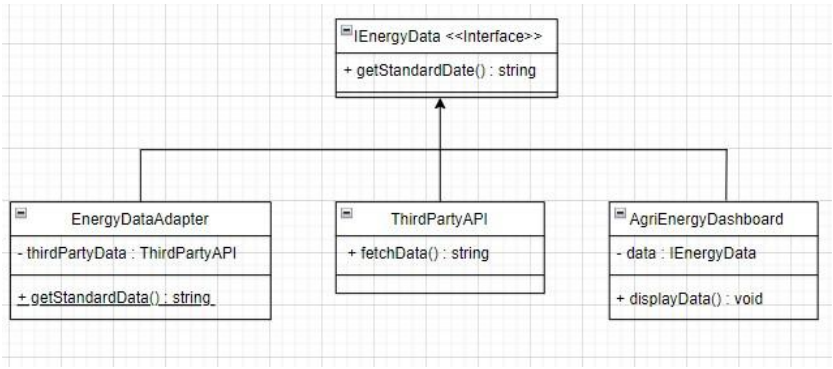

Figure 5: Adapter Pattern (GeeksForGeeks, 2025).

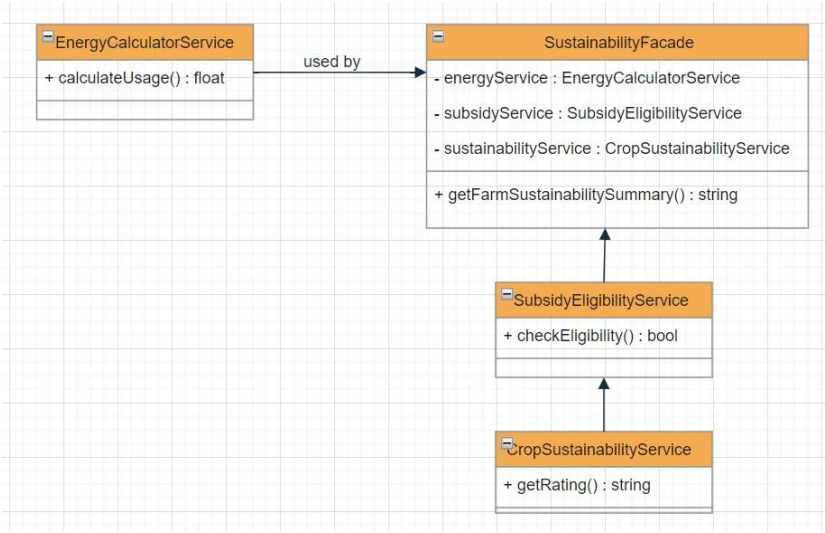

Figure 6: Facade Pattern (GeeksForGeeks, 2025).

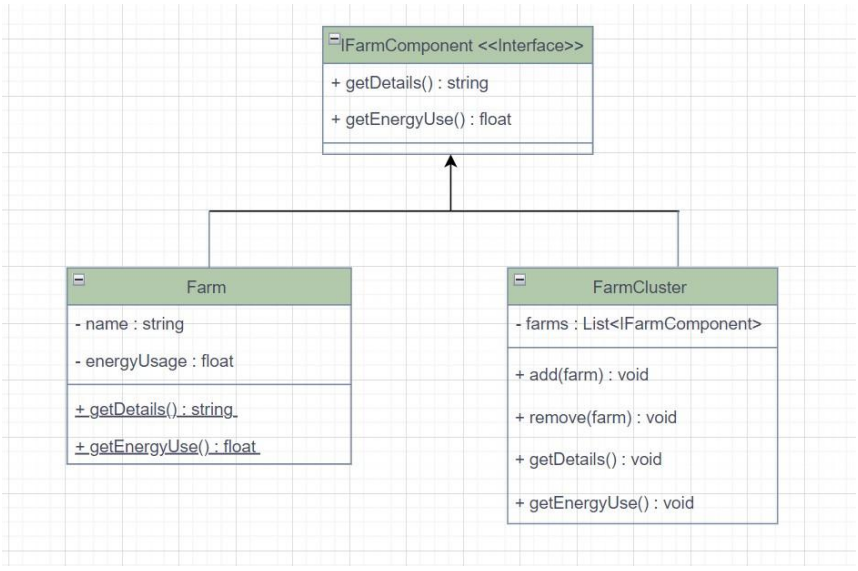

Figure 7: Composite Pattern (GeeksForGeeks, 2025).

Behavioural Patterns – Managing Communication Between Components	
Observer Pattern	Farmers might need real-time notifications about weather changes, equipment failures, or subsidy approvals. The Observer Pattern ensures that multiple users receive updates when relevant data changes. For example, a *WeatherObserver* listens for climate updates and automatically alerts registered farmers about upcoming storms.
Strategy Pattern	The platform may support multiple sustainability strategies, such as solar-powered irrigation, wind energy, or biofuel alternatives. A Strategy Pattern allows selecting the best approach dynamically. For example, a SustainabilityStrategy interface could allow different energy optimization techniques to be applied to a farm based on location, budget, and resource availability.
Command Pattern	The system may need to track and manage user actions, like applying for grants, submitting reports, or purchasing energy credits. The Command Pattern ensures that actions can be stored, undone, or repeated. For example, a *GrantApplicationCommand* allows farmers to apply for government subsidies, with the ability to cancel or modify the application before submission.

(Kumar, 2024).

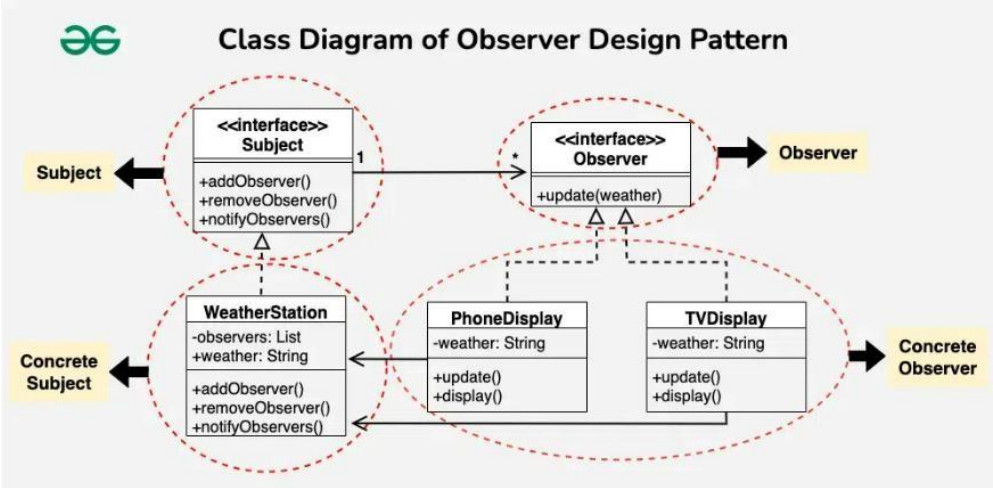

Figure 7: Observer Pattern (GeeksForGeeks, 2025).

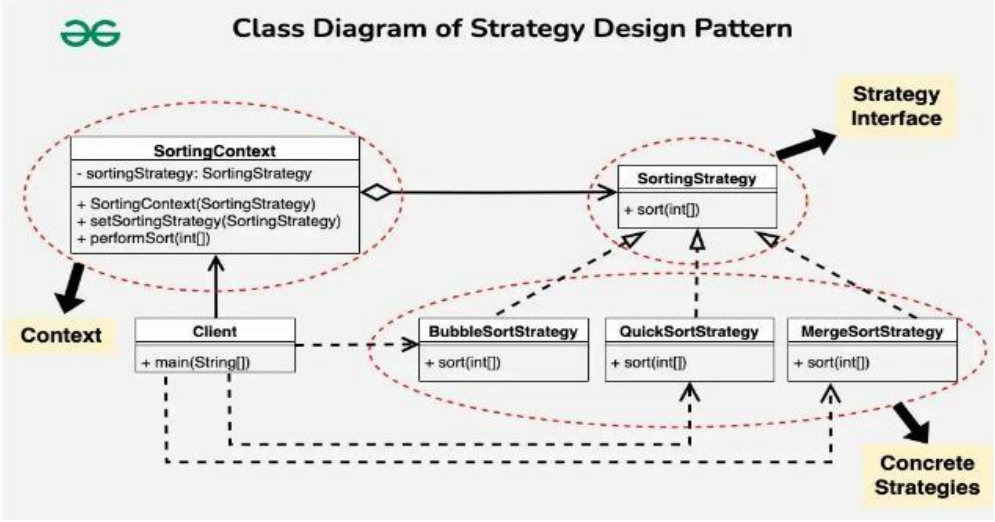

Figure 8: Strategy Pattern (GeeksForGeeks, 2024).

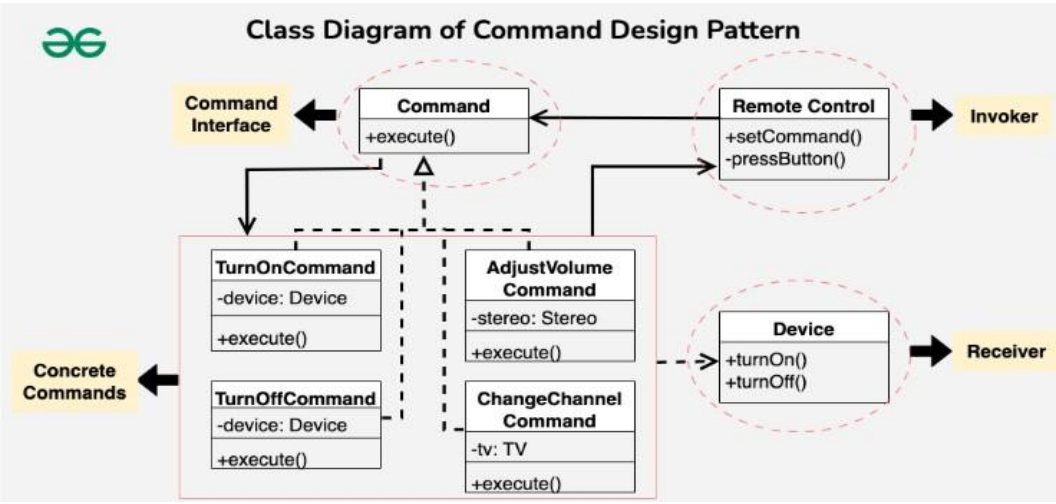

Figure 9: Command Pattern (GeeksForGeeks, 2025).

What are Architecture Patterns – Software architecture serves as the blueprint for a system, outlining how various components work together to meet user needs and business goals. It helps developers understand how to structure the software and its interactions to meet specific requirements. To guide the design, we use software architecture patterns, which are proven methods for organizing and connecting system components. These patterns are designed to solve common problems in software design and have been tested and refined over time (GeeksForGeeks, 2024).

Relevance of Architecture Patterns

Layered (N-Tier) Architecture	Ensures a structured and modular system by separating concerns (UI, business logic, data management), improving maintainability and scalability.

Client-Server Architecture	Supports Agri-Energy Connects web-based nature, providing centralized resource management and secure data exchange between farmers, energy experts, and stakeholders.
Microservices Architecture	Enables real-time updates and automated actions, such as alerts for energy production changes or notifications when surplus energy is available.
Event-Driven Architecture	Provides a clear separation of concerns in web-based applications, improving maintainability, testability, and user experience.
Model-View-Controller (MVC) Architecture	Facilitates seamless integration of various independent services within Agri-Energy Connect, enabling smooth interactions between farming tools, energy management systems, and sustainability analytics.
Repository Pattern	Enhances data management by abstracting database interactions, improving maintainability and ensuring efficient access to farm data, energy transactions, and collaboration records.
CQRS (Command Query Responsibility Segregation)	Optimizes system performance by separating data modification (commands) from retrieval (queries), ensuring smooth handling of large-scale real-time energy and agricultural data.
Domain-Driven Design (DDD)	Helps structure Agri-Energy Connect around core agricultural and energy management domains, ensuring the system aligns with business needs and sustainability goals.

(Morgan, 2023)

Integration of design and architecture patterns into the AgriEnergy Connect Platform

Design Patterns Integration		Reason
Singleton Pattern	The Singleton Pattern will ensure that certain critical components of the platform, such as the configuration manager, database connection context, or logging service, are instantiated only once and used globally throughout the application. For example, a single instance of a configuration manager will manage environment variables and system settings to ensure consistency and avoid conflicts across different modules or services. This approach promotes resource efficiency and centralized control (GeeksForGeeks, 2025).	**Centralized Control and Resource Efficiency** The Singleton Pattern is crucial for managing shared resources across the platform, such as configuration settings or logging services. In a platform with multiple modules marketplace, courses, projects, and forums having a single source of truth for configurations ensures consistency and avoids potential conflicts. It also promotes resource efficiency by reusing the same instance, reducing memory overhead and simplifying debugging when issues arise (GeeksForGeeks, 2025).
Factory Pattern	The Factory Pattern will be used to dynamically generate different object types based on input or context, especially when dealing with user roles or course categories. During user registration, the system will automatically generate the appropriate role object such as Farmer, Green Energy Provider, or Sustainability Expert without requiring the application logic to manually determine which class to instantiate. This approach ensures flexibility and simplifies the process of extending or updating roles and course types in the future (GeeksForGeeks, 2024).	**Supporting Extensibility and Flexibility** The Factory Pattern simplifies object creation and allows the system to easily adapt to different user roles and platform components. This is valuable in Agri-Energy Connect, where a wide range of users (e.g. farmers, sustainability experts, investors) require different access levels and functionalities. By using a factory to generate these rolebased objects dynamically, the platform ensures that future additions or modifications to user types can be handled with minimal code changes, supporting a flexible and scalable system (GeeksForGeeks, 2024).

Strategy Pattern	The Strategy Pattern will be implemented to allow multiple authentication methods without tightly coupling them to the login logic. Users will be able to log in using various methods such as email/password, Google Sign-In, or even future extensions like	**Promoting Modularity and User-Centric Access** With diverse users accessing the platform through different devices and preferences, the Strategy Pattern supports modular authentication options. It allows Agri-Energy Connect to integrate multiple login strategies (e.g. email,
	biometric authentication. Each login method will be encapsulated as a separate strategy, making it easy to plug in new authentication options or switch between them without altering the core application code (GeeksForGeeks, 2024).	Google, biometrics) without tightly coupling them to the login logic. This modularity ensures the platform can accommodate user needs while remaining secure and adaptable to evolving authentication trends especially important as security requirements and access preferences shift over time (GeeksForGeeks, 2024).
Observer Pattern	The Observer Pattern will enable real-time communication and responsiveness in the platform by notifying subscribers of updates or changes in the system. For instance, when a new collaborative project is created or updated, all involved participants and followers will automatically receive notifications. This promotes timely collaboration and enhances user engagement by keeping all stakeholders informed about relevant actions and updates within the platform (GeeksForGeeks, 2025).	**Enabling Real-Time Collaboration and Engagement** The Observer Pattern is ideal for a platform focused on collaboration and community engagement. It ensures that users are kept up to date with changes such as new discussion posts, project updates, or course completions. This real-time feedback loop increases platform interactivity and encourages user participation. In a collaborative ecosystem like Agri-Energy Connect, this pattern fosters a more connected and responsive user experience, improving engagement and trust among stakeholders (GeeksForGeeks, 2025).

Repository Pattern	The Repository Pattern will be integrated into AgriEnergy Connect to abstract and encapsulate all data access logic, creating a clear boundary between the data and the business logic. This promotes cleaner code, improves maintainability, and makes the application easier to test and modify in the future. For example, CRUD operations related to users, educational courses, and marketplace products will be managed through repositories, allowing seamless data interaction without exposing the underlying data source logic directly to the application layers (GeeksForGeeks, 2024).	**Enhancing Maintainability and Scalability** The Repository Pattern introduces a clean separation between the application's business logic and data access logic. This makes the codebase easier to maintain and extend, which is critical for a growing platform like AgriEnergy Connect. As new features such as additional user types, new course categories, or green products are added, the repository structure allows developers to make changes without disrupting the rest of the application. It also simplifies testing by allowing mock data layers to be injected during unit testing (GeeksForGeeks, 2024).

Architecture Patterns Integration	Reasons
Layered (N-Tier) Architecture	The system will be structured using the N-Tier architecture consisting of: ✞ Presentation Layer (ASP.NET MVC Views and Razor Pages) for user interfaces tailored to farmers, green energy providers, and sustainability experts. ✞ Business Logic Layer (Controllers and Services in C#) to handle workflows such as product comparison, training enrolment, or collaborative project management. ✞ Data Access Layer (using Entity Framework Core or ADO.NET) to manage database interactions with the SSMS SQL Server (Cherif, 2024).	This promotes separation of concerns, making the system modular and easier to maintain. UI components can evolve independently of business logic and data access. Developers can work in parallel across layers, enabling better team collaboration, easier bug fixing, and code reuse (Cherif, 2024).
Microservices Architecture	The platform will be modularized into independently deployable ASP.NET Core Web APIs, each representing a microservice, such as: + User Management Service + Green Energy Marketplace Service + Collaboration & Projects Service + Education & Training Module (GeeksForGeeks, 2024).	Each service can be developed, tested, and scaled independently. For instance, the Marketplace Service can scale during peak farming seasons without affecting the rest of the system. This enhances fault tolerance, supports CI/CD, and improves platform resilience and performance (GeeksForGeeks, 2024).

Client-Server Architecture	AgriEnergy Connect will follow a Client-Server model, where clients (web apps or mobile apps) interact with centralized servers via secure RESTful APIs built with ASP.NET Core (GeeksForGeeks, 2024).	Centralizing logic and data on the server ensures data consistency and security, while allowing users to access services from distributed clients. This is particularly important for rural access, where users may rely on lightweight front-ends with intermittent connectivity.
Event-Driven Architecture	Making use of event messaging for real-time interactions such as, Notifications for funding availability, Alerts when projects are updated, Energy usage threshold exceeded. This will be implemented with message queues like RabbitMQ or Kafka (GeeksForGeeks, 2024).	This pattern brings agility and responsiveness to the platform. Using event messaging (e.g. via RabbitMQ or Kafka), real-time alerts can be triggered when a new funding opportunity is available, a project milestone is reached, or an energy consumption threshold is crossed. This enhances user experience by keeping stakeholders informed and engaged without manual refreshes or polling. It also enables automated workflows, reducing the need for human intervention and increasing platform intelligence (GeeksForGeeks, 2024).
Model-ViewController (MVC)	Using ASP.NET Core MVC, the platform will cleanly separate: - Model (C# classes representing business entities and logic) - View (Razor Views for user interfaces) - Controller (handling user input and system responses) (GeeksForGeeks, 2024).	MVC architecture is ideal for the web-based parts of AgriEnergy Connect. It cleanly separates the presentation layer (View), business logic (Model), and user input handling (Controller), which improves code readability and modularity. By adopting frameworks like ASP.NET Core MVC or Django, the platform benefits from rapid development, easier testing, and smoother UI updates ensuring the web interface remains intuitive and aligned with evolving user needs (GeeksForGeeks, 2024).

In conclusion, this report has outlined the key requirements and design specifications for the Agri-Energy Connect platform, developed using Visual Studio C# (MVC) and SQL Server Management Studio (SSMS). These tools will enable us to create a scalable, secure, and userfriendly platform for Thaba Verde Farm, focusing on features like user management, the green energy marketplace, and training modules.

By leveraging Visual Studio C# and SSMS, the platform will be able to integrate modular, cloud-based architecture and responsive design, ensuring future scalability and efficient performance. The design is shaped by real-world farming scenarios, ensuring the platform meets Thaba Verde's sustainability needs from day one.

With this platform, Thaba Verde will have the tools to lead in sustainable farming and green energy solutions, driving innovation and collaboration for long-term impact. We are excited to bring this vision to life and support Thaba Verde in achieving its goals through a powerful digital platform.

References

Adams, K. M., 2015. *Non-functional Requirements in Systems Analysis and Design.* illustrated ed. Cham: Springer international publishing.

Cherif, Y., 2024. *Understanding the Layered Architecture Pattern: A Comprehensive Guide.* [Online] Available at: https://dev.to/yasmine_ddec94f4d4/understanding-the-layered-architecture-pattern-a-comprehensiveguide-1e2j [Accessed 5 April 2025].

Ciampa, M., 2021. *CompTIA Security+ Guide to Network Security Fundamentals.* 7th ed. Boston: Cengage Learning.

FasterCapital, 2024. *Scope Definition: How to Define the Scope of Your Enterprise Analysis Project.* [Online] Available at: https://fastercapital.com/content/Scope-Definition--How-to-Define-the-Scope-of-Your-EnterpriseAnalysis-Project.html [Accessed 2 April 2025].

GeekForGeeks, 2024. *What is Scalability and How to achieve it?.* [Online] Available at: https://www.geeksforgeeks.org/what-is-scalability/ [Accessed 3 April 2025].

GeeksForGeeks, 2024. *Client-Server Architecture - System Design.* [Online] Available at: https://www.geeksforgeeks.org/client-server-architecture-system-design/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *Event-Driven Architecture - System Design.* [Online] Available at: https://www.geeksforgeeks.org/event-driven-architecture-system-design/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *Factory method Design Pattern.* [Online] Available at: https://www.geeksforgeeks.org/factory-method-for-designing-pattern/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *Microservices Design Patterns.* [Online] Available at: https://www.geeksforgeeks.org/microservices-design-patterns/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *MVC Architecture - System Design.* [Online] Available at: https://www.geeksforgeeks.org/mvc-architecture-system-design/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *Repository Design Pattern.* [Online] Available at: https://www.geeksforgeeks.org/repository-design-pattern/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *Strategy Design Pattern.* [Online] Available at: https://www.geeksforgeeks.org/strategy-pattern-set-1/ [Accessed 5 April 2025].

GeeksForGeeks, 2024. *Types of Software Architecture Patterns.* [Online] Available at: https://www.geeksforgeeks.org/types-of-software-architecture-patterns/ [Accessed 3 April 2025].

GeeksForGeeks, 2025. *Functional vs. Non Functional Requirements.* [Online]

Available at: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/
[Accessed 2 April 2025].

GeeksForGeeks, 2025. *Observer Design Pattern.* [Online]
Available at: https://www.geeksforgeeks.org/observer-pattern-set-1-introduction/
[Accessed 5 April 2025].

GeeksForGeeks, 2025. *Singleton Method Design Pattern.* [Online]
Available at: https://www.geeksforgeeks.org/singleton-design-pattern/
[Accessed 5 April 2025].

GeeksForGeeks, 2025. *Software Design Patterns Tutorial.* [Online]
Available at: https://www.geeksforgeeks.org/software-design-patterns/
[Accessed 3 April 2025].

Grow Solutions, 2023. *Functional and Non-Functional Requirements: The Ultimate Checklist with Examples.* [Online]
Available at: https://medium.com/@growsolutions/functional-and-non-functional-requirements-the-ultimate-checklistwith-examples-cde16aba33d7 [Accessed 2 April 2025].

Information Services, n.d. *Non Functional.* [Online]
Available at: https://www.requirements.co.za/non-functional/#section-2-2 [Accessed 3 April 2025].

IPH Technologies, 2024. *Top 10 Tips for Optimizing Software Performance.* [Online]
Available at: https://iph-technologies.medium.com/top-10-tips-for-optimizing-software-performance-cf4f55ed8f9c
[Accessed 3 April 2025].

Jaiswal, S., 2024. *Medium.* [Online]
Available at: https://medium.com/@satyendra.jaiswal/understanding-layered-architecture-a-comprehensive-guide4c2eee374d18
[Accessed 5 April 2025].

John W. Satzinger, R. B. J. S. D. B., 2015. *Systems Analysis and Design in a Changing World.* 7th ed. Boston: Cengage Learning.

Kumar, S., 2024. *Design Patterns - Definition, Motivation & Classification.* [Online]
Available at: https://www.scaler.com/topics/software-engineering/design-patterns-in-software-engineering/ [Accessed 4 April 2025].

Llyukha, V., n.d. *The Key Difference Between Functional vs. Non-Functional Requirements.* [Online]
Available at: https://jelvix.com/blog/functional-vs-nonfunctional-requirements [Accessed 2 April 2025].

Morgan, N., 2023. *What Are The 10 Most Common Software Architecture Patterns?.* [Online]
Available at: https://medium.com/@the_nick_morgan/what-are-the-10-most-common-software-architecture-patternsfaa4b26e8808
[Accessed 3 April 2025].

Sommerville, I., 2016. *Software Engineering.* 10th ed. Harlow : Pearson Education.