# PROG 7311

## POE PART 1

Sajana Bidesi
ST10249843

# Table of Contents

# Non-Functional Requirements

## 1. Scalability

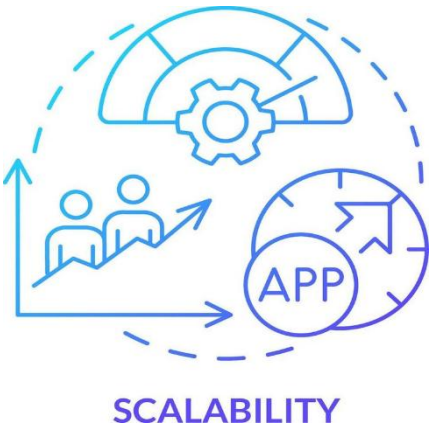


*Figure 1Scalability Adapted from Pinterest By Creative Design Assets (Creative Design Assets, n.d.)*

**What is scalability?**

Scalability is assessing a system's **ability to handle growing users, volume data** (Watson, 2020), and increased workloads without compromising the systems **performance** and user experience (Watson, 2020) with minimum changes made to the system (AltexSoft, 2022).

**How is scalability a critical non-functional requirement for the success of Agri-Energy Connect**

For Agri-Energy Connect, scalability will be used to handle the growing number of farmers, green energy experts, enthusiasts, and employees on the system whilst adopting tools for optimized collaboration features, resource sharing, marketplace, education and training, project collaboration, and funding opportunities

**How scalability shapes the planning and execution of Agri-Energy Connect:**

**Planning phase:**

- Determine **expected user load** of farmers, energy providers, etc.
- Identify the **features** that would have **high traffic** and data such as discussion boards, forums, product reviews, and educational services.
- Design a **scalable architecture** (ByteByteGo, 2024b) using modular components and microservices.
- Makes use of **database scalability** to manage large data loads such as database sharding or replication (Oracle, 2021).
- The adoption of **microservices architecture** (altexsoft, 2022) for independently deployable and scalable features. This will allow for separate development, deployment, and scaling and assists with reducing interdependencies and improve maintainability.
- Adopting **cloud technologies** (Faruqui, 2024) for scalable and flexible infrastructure such as AWS or Azure that support **auto scaling and load balancing** with **automatic deployment** (Dwyer, 2023).
- Choose a **relational database** system (MySQL, Oracle Db, etc.) (Oracle, 2021).

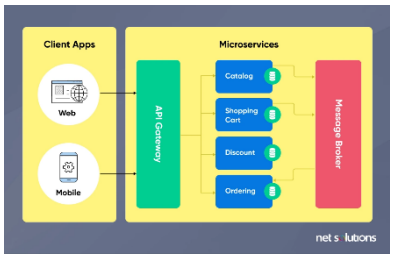


*Figure 2 What are Microservices? Adapted from (Singla, 2023)*


*Figure 3 Cloud computing: Everything you need to understand about the cloud. Adapted from (TechGiga, 2020)*

**Execution phase:**

- Develop the platform using **stateless services** to enable easier scaling across multiple servers (ByteByteGo, 2024b). This avoids storing session data on the server while enabling horizontal scaling to allow for load balancing. This will improve resilience and recovery should the system fail.
- Optimise **backend processing and APIs** (altexsoft, 2022) to ensure they can handle the increased loads.
- Define database schema with **normalised tables** to store user data, product listings, funding data, etc (Oracle, 2021).
- Integrate **pagination** (Khatri, 2025), **lazy loading** (Khatri, 2025), and **asynchronous processing** to manage data most efficiently.
- Prioritises **performance testing** (Doshi, 2023), load balancing, automated scaling, and stress testing to simulate heavy usage on the system and identify **bottlenecks**.
- Requires **automated deployment** (AWS, n.d.) to support fast updates and handle user growth, make use of **containerisation tools** (RedHat, 2022) for flexible scaling.
- Ensures the platform remains **responsive** and stable as the growing demand of users and data increases.
- Maintain optimal performance by ensuring **horizontal scaling** (database sharding) (AWS, n.d.).
- Utilise **microservice architecture** (Swimm Team, 2024) to break the system into **independently deployable services** (Swimm Team, 2024) for forums, training materials, chat/collaboration, and product listings.


*Figure 4 Deployment Automation Adapted from (Hornay, 2021)*

## 2. Security

**What is security?**

Security NFRs assess how the data in a system are **protected** against unauthorised access, security breaches, and cyberattacks. To prevent any of this, the system needs to ensure implementation of data encryption, authentication, and authorisation mechanisms to protect users' personal information, financial information, etc. to maintain user satisfaction and trust (Akhtar, 2023).

**How is security a critical non-functional requirement for the success of Agri-Energy Connect**

For Agri-Energy security protocols such as **role-based authentication**, ensuring that the real-time collaboration on products is private, ensuring financial transactions are secure, user data is encrypted, the proprietary farming techniques, and the educational views for specific users who have access to it. These all ensure that there is no violation of any of the **privacy regulations** such as POPIA.

**How security shapes the planning and execution of Agri-Energy Connect:**

**Planning phase:**

- Identify **security measures** as the system will have to handle users' personal information, financial transactions, and proprietary farming techniques.
- Assess **potential threats** such as cyberattacks, data breaches, and unauthorised access to the system.
- Ensure data protection complies with **privacy regulations** e.g. POPIA (POPIA, 2021).
- Integrating **security-by-design principles** (Kirvan, n.d.) to embed security throughout the system from the beginning.
- Design the platform with features such as **role-based access control (RBAC)** (Lindemulder and Kosinski, 2024), encrypted communications between users, and secure data storage. This will restrict user access depending on their roles example: farmers, green energy experts, and enthusiasts. This will enhance security and simplify the user management and in turn improve the user experience by displaying separate dashboards or features for each of them.
- Plan for **secure APIs**.
- Ensure **separation between frontend and backend systems**.

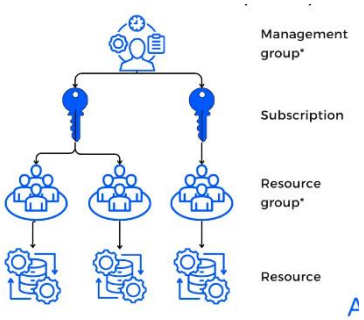


*Figure 5 Role-based access control (RBAC) in 2025 Adapted from (Dilmegani, 2025)*

**Execution phase:**

- Apply **secure coding standards** (Foster, 2020).
- Implement **multifactor authentication (MFA)** (AWS, 2023) for user accounts and admin access.
- Limit **user privileges** based on their roles to reduce risk exposure.
- Conduct **penetration testing** (Doshi, 2023) and apply **vulnerability assessments** to identify and fix security vulnerabilities.
- Perform **code reviews** (Doshi, 2023) to fucus on security and vulnerabilities.
- Run **privacy and security audits** (Doshi, 2023) to ensue sensitive data is appropriately protected.
- Use **HTTPS** (Cato Networks, 2024)**, firewalls** (Cato Networks, 2024)**, and secure hosting environments** (Cato Networks, 2024) to protect data.
- Regularly **update software** and monitor system for suspicious activity.

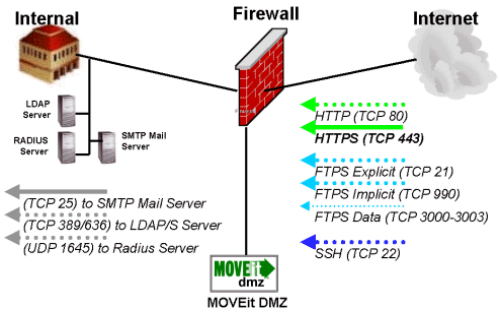


*Figure 6 System Configuration - Firewall Configuration Adapted by (IP Switch, 2025)*

## 3. Usability

**What is usability?**

Ensuring that the system is **intuitive** and user-friendly is how usability is measured. This includes aspects such as the **user interface design** (Akhtar, 2023), **accessibility** of the system, and the **user experience** gained. The higher usability of a system is to ensure that the system is **easy to use** (Akhtar, 2023), where users would not need assistance with navigating the system. The **5 e's of usability** (Komninos, 2016) are **effective, efficient, engaging, error tolerant, easy to learn**. These ensure that the development team understands the user needs, evaluate the product, and improve user experience (Komninos, 2016).

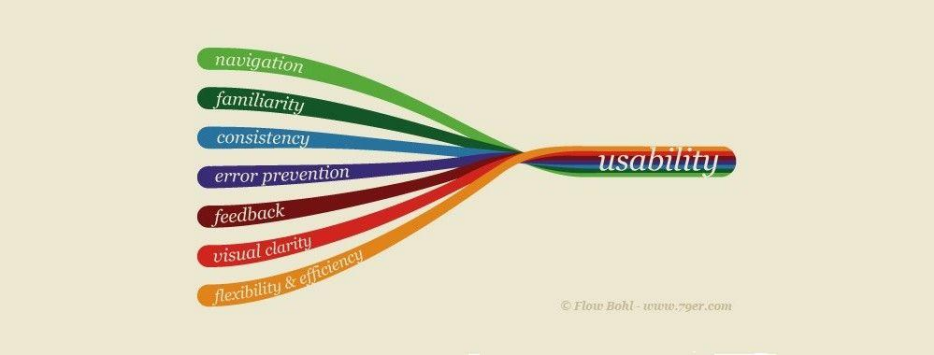


*Figure 7 Features of Usability Adapted from (Komninos, 2016)*

**How is usability a critical non-functional requirement for the success of Agri-Energy Connect**

For Agri-Energy Connect, the usability must ensure that the design must be user-friendly, intuitive, and accessible to users with **varying levels of technological expertise**. The system must include using **simple language** and **infographics**, clear navigation, and a responsive design. The discussion boards should be **legible** and easy to read, the marketplace should easily allow users to compare and learn the different technologies for green energy, the online educational and training services should have a progress dashboard to track progress in webinars or training modules, project collaboration should include a real-time chat and video calling integration, there should be **icons** to be able to distinguish between the **navigations** sections of the system.

**How usability shapes the planning and execution of Agri-Energy Connect:**

**Planning phase:**

- Identify the needs and capabilities of the **diverse users** of the system.
- Ensure that the platform is **inclusive** and **intuitive** for all user types of varying technical expertise (Yalanska, 2020).
- Plan for a **clean and simple interface** with easy navigation using user flows and interface wireframes (Yalanska, 2020).

**Execution phase:**

- Build user interfaces with **consistent layouts** with simple and helpful tooltips (Yalanska, 2020).
- Use **iconography, typography** (Yalanska, 2020) and visual cues to guide the users rather than having text everywhere.
- Conduct usability testing with real users to evaluate design and navigation choices. Gather feedback from user testing on navigation, readability, and user satisfaction.

## 4. Performance

**What is performance?**

Performance identifies how **quick** a system **responds** to the users' actions under a workload. The **performance metric** (altexsoft, 2022) explains how long it takes before an operation takes place by the users in the system. This may not only be the frontend (altexsoft, 2022), it may also take place in the backend such as backups or system refactoring (Design Gurus, 2024).

**How is performance a critical non-functional requirement for the success of Agri-Energy Connect**

In Agri-Energy Connect, performance is the ability to **support real-time data sharing** between farmers, green energy experts, educational and other users across the sustainable farming hub, green energy marketplace, and project collaboration. When users host forums, discussion boards, educational classes, comparing green technologies, and/or manage funding proposals, it must allow **seamless user interaction, low latency** during peak usage, and ensuring **quick system response time** under high user load.

**How performance shapes the planning and execution of Agri-Energy Connect:**

**Planning phase:**

- Define **performance benchmarks** (Paredes, 2023). Examples:
    - The page should load within 2 seconds.
    - The system should support at least 2000 users at a time.
    - The system must maintain 99.9% uptime.
    - Educational videos should have a buffering delay of less than 3 seconds.
    - Funding proposals or collaboration should submit within 3 seconds.
    - The search filter in the marketplace should load results within 2 seconds.
- Identify which **activities or microservices** would require **faster performance** such as accessing the marketplace, submitting funding proposals, or streaming educational content (Swimm Team, 2024).
- Choose an infrastructure that supports **fast processing speeds**, low latency, and efficient load balancing (Scale Computing, 2023).
- Plan for **caching strategies** for efficient server response times (Emerich, 2023).
- Select **high performance database optimisation** for read/write operations (Oracle, 2021).

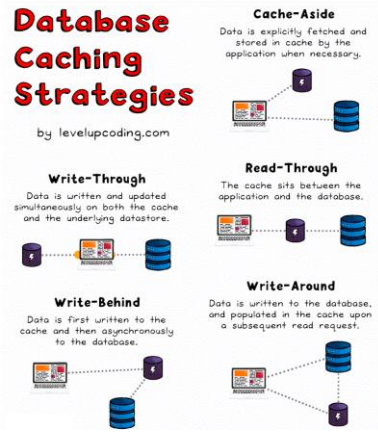


*Figure 8 Caching strategies adapted from (Ganesh, 2024)*

**Execution phase:**

- **Compress images**, integrate **lazy loading** (Verpex, 2023), and **minify files** to **optimise front-end code** (Imperva, 2019).
- Ensure back-end processing and **queries** for the database are **efficient** and **indexed** for fast data retrieval.
- Use **pagination** for **displaying large volumes of data** such as product listings, project collaborations, discussion boards, green energy marketplace, etc. (Verpex, 2023).
- Conduct **performance testing** (stress, load, spike) to evaluate behaviour under different system conditions.
- Monitor **bottlenecks**, **latency**, and **server health** throughout development and deployment (AWS, n.d.).
- Implement **auto-scaling** to maintain consistency with the performance during peak usage of the system (AWS, n.d.).

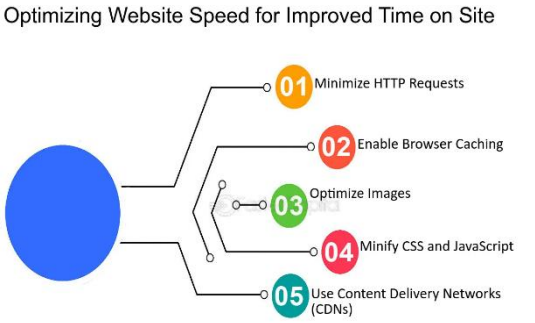


*Figure 9 Optimising performance on a website Adapted from (Faster Capital, 2024)*

## 5. Reliability

### What is reliability?

Reliability ensures the system would **operate consistently**, and **reliably** under specific conditions (Barrett, 2023) and includes factors such as **mean time to failure (MTTF)**, **mean time to repair (MTTR)**, and **availability** (Saifi, 2023). It must ensure system availability - how often the system must be functional without crashing (Barrett, 2023), **fault tolerance** – how the system recovers after a fail without any data loss or downtime (Barrett, 2023), **error handling** – the system must be able to handle errors by sending informative messages regarding the error that takes place as well as log errors to troubleshoot future issues that may arise (Barrett, 2023).

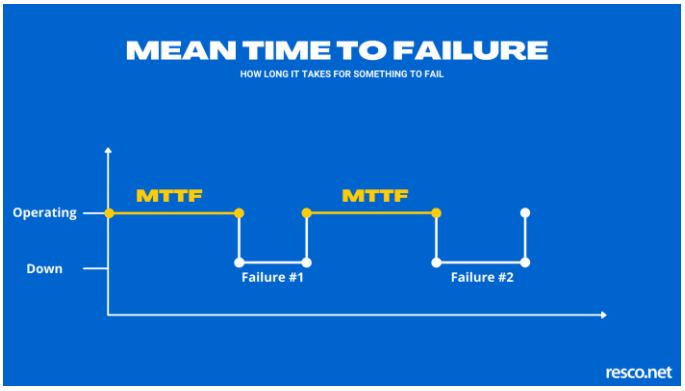


*Figure 10 MTTR diagram Adapted from (RescoNext, 2025)*

### How is reliability a critical non-functional requirement for the success of Agri-Energy Connect

The reliability requirement for Agri-Energy Connect ensures that the system remains responsive and **consistently operational** and available to users (farmers, green energy experts, and other stakeholders), all **without unexpected downtime**. Since the system has multiple functionalities such as the sustainable farming hub, green energy marketplace, sharing or resources for best farming practices, funding opportunities, and educational materials the platform must maintain a **high uptime** and must have a **robust backend** to manage **error handling**. This is important for **maintaining trust** with the users, and support continuous project collaboration, grant applications and educational services.

**How reliability shapes the planning and execution of Agri-Energy Connect:**

**Planning phase:**

- Set **clear targets** such as the uptime (Wright, 2023), fail-safe mechanisms (Full Stack Developer, 2024), and error recovery capabilities.
- Identify **features that must remain** reliable under all conditions such as payment processing, proposal submissions, account management, etc.
- Design an **architecture** (ByteByteGo, 2024b) **with backup services and failover systems** to handle system crashes.
- Decide of a **cloud platform that has a built-in reliability feature** and SLAs (Faruqui, 2024).
- Incorporate **database replication and automated backups** to preserve data integrity (Oracle, 2021).

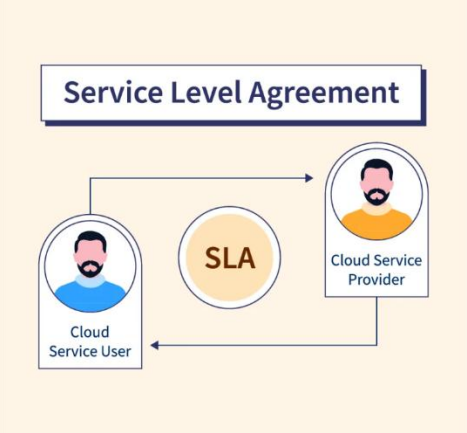


*Figure 11 SLA diagram Adapted from (Ali, 2024)*

- Conduct **risk analysis** (Doshi, 2023) to identify the potential risks that may be caused in the system and potential failure points such as power outages or connectivity issues.
- Prepare **disaster recovery plans** (Doshi, 2023) and incident response strategies to minimise data loss.

**Execution phase:**

- Use **error handling mechanisms** (Doshi, 2023).
- Implement **logging and monitoring** to detect, record, and respond to failures.
- **Disaster recovery drills** will validate backup and restore processes (Doshi, 2023).
- Set up **automated failover systems** (Doshi, 2023).
- Ensure **regular updates** take place.

## Mapping of non-functional requirements to platform features and implementation strategies.

| Non-Functional Requirement | Platform Features | Tools/Strategies Used |
|---|---|---|
| **Scalability** | • Discussion boards<br>• Green energy marketplace<br>• Educational content<br>• Project collaboration | • Microservices architecture<br>• Cloud infrastructure (AWS, Azure)<br>• Automatic scaling, deployment, and load balancing<br>• Database sharding and replication<br>• Stateless services |
| **Security** | • User authentication and access control<br>• Financial transactions<br>• Proprietary data protection such as farming techniques | • Rol-based access control (RBAC)<br>• Data encryption<br>• Secure APIs<br>• POPIA compliance<br>• Firewalls, HTTPS, vulnerability assessments |
| **Usability** | • User interface and navigation<br>• Discussion forums<br>• Educational dashboards<br>• Marketplace comparison tools | • Simple and clean UI design<br>• Tooltips and icons<br>• Responsive layout<br>• Usability testing and feedback |
| **Performance** | • Marketplace browsing<br>• Real-time collaboration tools<br>• Educational content streaming<br>• Funding proposals submissions | • Caching strategies<br>• Pagination and lazy loading<br>• Code minification<br>• Performance testing (stress, load, spike)<br>• Backend query optimisation |
| **Reliability** | • Account and user data management<br>• Payment processing<br>• Submission of proposals<br>• System uptime for forums and webinars | • Automatic failover systems<br>• Backup and restore mechanisms<br>• Real-time monitoring and logging<br>• Disaster recovery plans<br>• Regular software updates |

# Role of Design and Architecture Patterns in the Agri-Energy Connect Platform

## What are design patterns?

Design patterns are **toolkit and tested solutions** (Refactoring Guru, 2014) that **solve consistent occurring problems** in software development. They are **customisable, generic concepts** (Refactoring Guru, 2014) that are implemented specifically and used for solving a particular problem, meaning that the code of the same pattern applied to different programs, may be different (Refactoring Guru, 2014). Design patterns can be **categorized into three sections** for their purpose, these include creational patterns, structural patterns, and behavioural patterns (Refactoring Guru, 2014).

- **Creational patterns:** this pattern provides **mechanisms for object creation**. This pattern will increase the flexibility and reusability of the code (Refactoring Guru, 2014).
- **Structural patterns:** this pattern explains **how the object and classes in larger structures are assembled**, ensuring that the solution is still flexible and efficient (Refactoring Guru, 2014).
- **Behavioural patterns:** This pattern allows for the **assignment of responsibilities** between objects in software development project, as well as ensure **effective communication** within the project (Refactoring Guru, 2014).

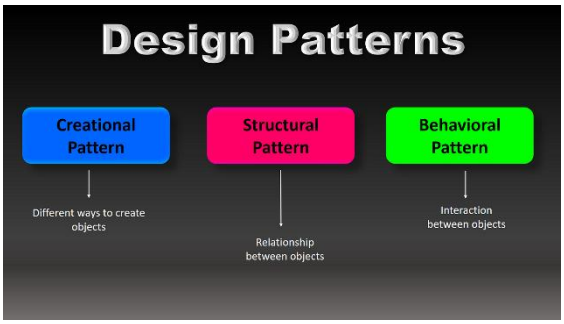


*Figure 12 Design patterns. Adapted from (Cheth Virajini, 2023)*

## What are architecture patterns?

Architectural patterns provide a **methodical approach** to solving recurrent design problems (ByteByteGo, 2024a). To address these design challenges, this pattern provides a **reusable approach** allowing them to be used across various projects (ByteByteGo, 2024a). Making use of architectural patterns allow for:

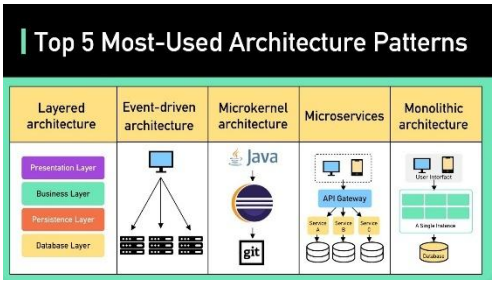


*Figure 13 Examples of architecture patterns. Adapted from YouTube at (ByteByteGo, 2023)*

- **Increased productivity:** developers can **rely on existing architecture patterns** enabling them to **save both time and effort** when developing a platform/program (ByteByteGo, 2024a). in doing so, developers can spend more time solving the problem instead of reinventing new solutions (ByteByteGo, 2024a) to solve the recurring issues.
- **Improved quality of code**: each of these patterns have a **specific coding standards** that need to be followed, in doing so, this will assist with ensuring that the code is **scalable, maintainable, and simple to understand** by other developers working on the project (ByteByteGo, 2024a).
- **Better communication:** To **improve communication and discussion** between developers, architecture patterns have **names, specific to their purpose**, for better understanding (ByteByteGo, 2024a).
- **Quicker development cycles:** relating to the productivity improvement, since the architecture patterns are already existent, developers would spend **less time developing new solutions**, and rather focus on the project at hand, allowing for a faster development process (ByteByteGo, 2024a).

**Relevance and value added to Agri-Energy Connect**

**Yes**, design patterns and architecture patterns are relevant for the development of the Agri-Energy Connect platform as they offer many benefits as mentioned above.

Design patterns are relevant as they provide **mechanisms for object creation, object responsibility assignment, and object integration** with classes in large scale projects, making the development process much simpler since most of the **code can be reused**, reducing redundancy and allowing for a more **scalable, maintainable, and secure system** (Refactoring Guru, 2014). Some examples of design patterns that would be relevant for Agri-Energy Connect would be **Factory Method** (as a creational pattern), **Façade** (as a structural pattern), and **Template** (as a behavioural pattern) (Refactoring Guru, 2014).

As for architecture patterns, these have multiple benefits that make them relevant for the development of Agri-Energy Connect, such as the **increased productivity, and quicker development cycles** (ByteByteGo, 2024a) as it is a reusable solution making the production and development a much smoother process. Architecture patterns such as the **layered architecture** which **separates concerns** to **improve structure and security**, **and microservices architecture** which allow features of the platform to **work independently** improving the **scalability** of the system (ByteByteGo, 2024a).

The platform must **accommodate for the diverse features** of Agri-Energy Connect such as the sustainable farming hub, green energy marketplace, educational and training sectors, project collaboration, and funding opportunities all whilst ensuring that the **backend framework remains robust and unified**.

**How design and architecture patterns would be integrated into Agri-Energy Connect.**

**Design Patterns:**

1. **Factory Method – Creational**

The creational category of design patterns is used for the creation of objects **without specifying their class types**, the factory method ensures that the **subclass takes the responsibility of object creation** which allows for better flexibility and extensibility (Shah, 2024). It entails using an interface for **creating objects in a superclass**, afterwards, allowing the **subclass to alter the types of objects created** (Refactoring Guru, 2014) by **encapsulating** (Shah, 2024) the object creation logic and **decoupling** (Shah, 2024) it from the client code. This provides a **centralized point of control for object creation**. The components required for the factory method are **Product, ConcreteProduct, Creator, and ConcreteCreator.**

This can be implemented into the Agri-Energy Connect platform for the **role-based authorisation**. This pattern enables the creation of different types of user objects such as farmers, green energy experts, enthusiasts, etc.

**Example:**

In Agri-Energy Connect, there are multiple user roles with **distinct permissions and interfaces** granted to each:

- **Farmers:** Access the marketplace, discussion forums, education and training, funding opportunities, add new products.
- **Green energy experts:** access the green energy marketplace, educational content, and project collaboration.
- **Enthusiasts:** discussion boards, sustainable farming hub

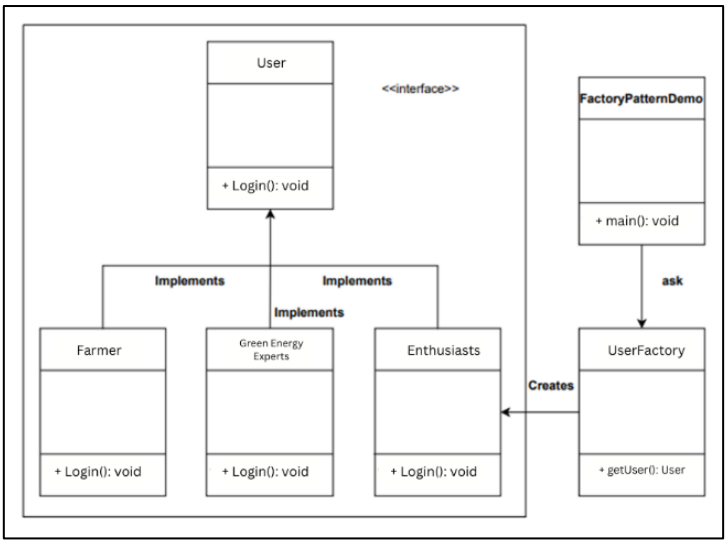


*Figure 14 An example of how the factory method would function for Agri-Energy Connect Created by Sajana Bidesi. Original image adapted from (Devduni, 2022)*

This method can generate the **appropriate user role object depending on their login crede**ntials. This is important as **each role will require different access privileges and interfaces**, and if a **new type of user** is to be implemented at a later stage, the addition can be completed **without rewriting code or existing logic**, the developer would simply have to **create a new subclass**. This ensures that **security, usability, performance, and reliability** is maintained by preventing unauthorised access, or unnecessary user experience issues.

## 2. Façade – Structural

The Façade pattern is a structural design pattern that **assists with complexity** and **improves system organisation**. It displays a **simplified unified interface** of a **complex system** increasing the usability (Kumar, 2023). This pattern involves one class which will provide simplified methods required by client who calls to methods of existing classes. The methods called are the ones that enable the interface to be simplified, ensuring no user is left confused on how to navigate the system. Agri-Energy Connect will grow **complex** with all the **interconnected subsystems and microservices**, such as the sustainable farming hub, green energy marketplace, educational subsystem, learning material, project collaboration pages, and proposal submissions for funding opportunities, and that could lead to many **challenges**. To solve the issues of difficult modifications and system maintenance, clustered client code, reduced code reusability (Dinidu Sachintha, 2025), etc. façade would be able to:

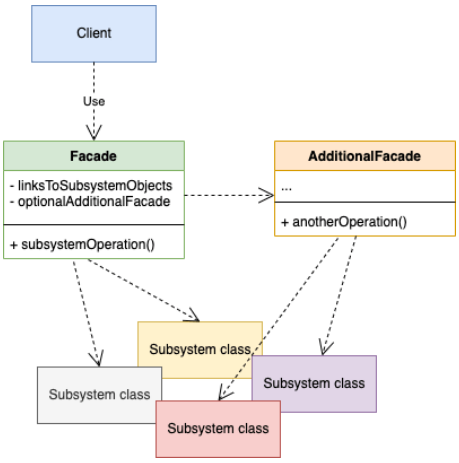


*Figure 15 How Facade works. Adapted from (LearnCsDesign, 2022)*

- **Subsystem Complexity: separates the clients** from the complex subsystem components (Dinidu Sachintha, 2025).
- **System Decoupling:** ensures that the system is more **maintainable** by **promoting loose coupling** between clients and subsystems (Dinidu Sachintha, 2025).
- **Layer Definition:** defines a **clear definition of layers in the application architecture** and improve the organization of the system (Dinidu Sachintha, 2025).
- **Legacy System Integration: older systems** would be able to be used easily as façade provides a **modern interface to the legacy code** (Dinidu Sachintha, 2025).

Using Façade will improve code maintainability, readability, and overall system architecture whilst ensuring that the user experience is not negatively impacted.

### 3. Template Method - Behavioural

The template method is a behavioural design pattern that **defines the structure of an algorithm** in the **superclass or abstract class** and lets the **subclass override the specific steps** without breaking the structure of it. To implement the template method the developer will have to create an abstract class and a concrete subclass (Jurran, 2024). The abstract class will contain the **basic workflow for the functionality** of the system while the concrete subclasses will have **overriding methods for the specific implementations**. The benefits are that the code will be **robust** and **secure**, the developer can let the client override only certain parts of the algorithm, as well as **deduplication** (Jurran, 2024).

In Agri-Energy Connect, this can be applied to **standardise workflows** (Source Making, 2025) such as applying for funding proposals, publishing a discussion post, or completing training modules. Here, the process is similar and remains constant but the steps change depending on the type of user on the system that access to these features.

**The abstract base class would be the template for:**

- Filling out the project proposal details
- Attaching supporting documentation
- Validating user information
- Submitting to gain the funding opportunities
- Sending a confirmation message
- Submitting a discussion on the forum

**The subclasses would contain the customised steps such as:**

- A farmer may be able to attach images of reports for their yield of crops throughout the seasons, submit on the discussion boards for advice, or view the boards to read on other farmers experience and be able to gain innovative insight, submit a proposal for a funding opportunity, or view green energy marketplace products.
- A green energy expert would be able to submit their green technology to the marketplace or be able to view and compare products.
- An enthusiast would be able to view the educational and training subsystem

**Architecture Patterns:**

1. **Layered Architecture (MVC)**

The Model View Controller pattern divides an application into three interconnected components:

- **Model:** Represents the data and business logic of the application. It directly manages the data, logic, and rules of the application.
- **View:** Handles the presentation and display of data. It provides the user interface and ensures that the data is presented in a user-friendly format.
- **Controller:** Manages user input and interactions. It acts as an intermediary between the Model and View, processing input, updating the Model, and selecting the appropriate View for output.

**The benefits of using MVCs layered architecture in Agri-Energy Connect**

**Failure Mitigation**

MVC's architecture helps in failure mitigation by isolating different components. If a failure occurs in one part of the system (e.g. the Model), it should not affect the other components (the View or Controller). This isolation makes it easier to diagnose and address issues without disrupting the entire application.

**Ease of Use**

MVC enhances ease of use by providing a structured framework for developing web applications. The separation of concerns simplifies the development process, allowing developers to work on individual components independently. Additionally, the MVC pattern supports a clean and intuitive user interface, making the application more accessible to users.

**MVC Scalability**

The MVC architecture is highly scalable, allowing applications to grow and adapt to increasing user demands. By separating the application into distinct components, MVC facilitates efficient management of expanding features and functionalities. This scalability is particularly beneficial for applications like Agri-Energy Connect that may evolve over time.

**Technology stack**

The MVC framework employs a technology stack comprising HTML, CSS, C#, and JavaScript, each playing a critical role in the functionality and user experience of Agri-Energy Connect Platform.


*Figure 16 3-Tiered architecture. Adapted from (Velu Dhanesh, 2018)*

**HTML:** Provides the foundational structure and content of web pages in Agri-Energy Connect, ensuring that information is organized and accessible. It defines how content is displayed to users (Velu Dhanesh, 2018).

**CSS:** Styles and designs the appearance of Agri-Energy Connect web pages, creating a user-friendly and visually appealing interface. It enhances the overall look and feel of the application (Velu Dhanesh, 2018).

**JavaScript:** Manages client-side interactions and dynamic content updates, adding interactivity to the Agri-Energy Connect web pages. It improves user experience by enabling real-time updates and responsive features (Velu Dhanesh, 2018).

**C#:** Handles server-side logic and interactions in Agri-Energy Connect. It manages business logic, data processing, and communication with the database, ensuring the smooth operation of the application (Velu Dhanesh, 2018).

This integration of HTML, CSS, C#, and JavaScript within the MVC framework ensures that Agri-Energy Connect is a robust, scalable, and user-friendly application designed to handle complex claim processing tasks efficiently (Velu Dhanesh, 2018).

The combination of these technologies within the MVC framework ensures that Agri-Energy Connect is a robust, scalable, and user-friendly application capable of efficiently handling complex claim processing tasks.

## 2. Microservice architecture

Microservices are **independent and loosely coupled** and usually **created and maintained by a smaller team of developers** (EdPrice-MSFT, 2023). Each microservice contains a **separate codebase, framework, storage, and design patterns**. Unlike monolithic architecture where the whole system is created and deployed at once, microservices should be able to **deploy independently** without having to rebuild and deploy the whole application, **reducing any errors** in the code (EdPrice-MSFT, 2023). Different development teams would be able to develop the separate microservices, these would include having a team for developing the sustainable farming hub, developing the green energy marketplace, the educational and training subsystem, and the project collaboration and funding opportunities. Using an **API gateway** would make the system run more **efficiently and effectively** (EdPrice-MSFT, 2023), as it would provide a **single-entry point** for users. So, instead of calling each service separately, clients would call to the API gateway and that would be able to **call the appropriate service on the backend** (EdPrice-MSFT, 2023). The API gateway would be able to handle **authentication, logging, SSL termination, and load balancing** (EdPrice-MSFT, 2023) making this architecture pattern more effective and scalable.

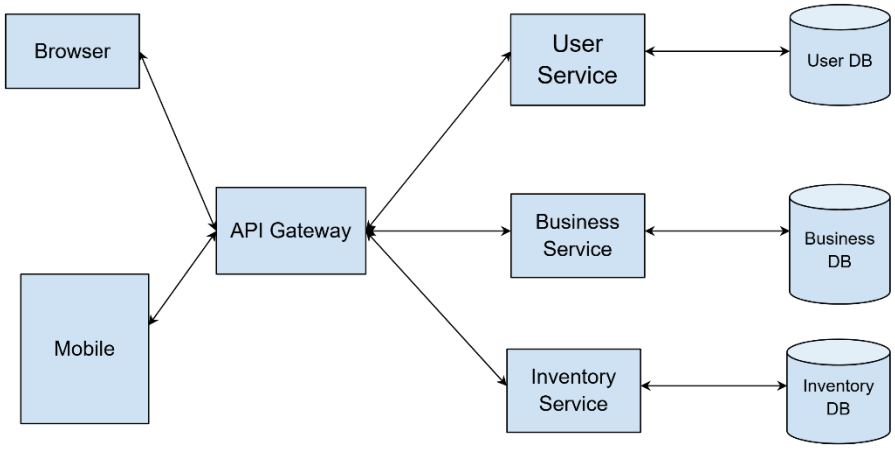


*Figure 17 Representation of a microservice architecture. Adapted from (Ahuja, 2021)*

# Table of figures

# Reference List


Ahuja, M., 2021. *What Are Microservice Architectures?* [online] Metricfire.com. Available at: <https://www.metricfire.com/blog/what-are-microservice-architectures/ > [Accessed 3 April 2025].

Akhtar, H., 2023. *Types of Non-Functional Requirements Examples*. [online] BrowserStack. Available at: <https://www.browserstack.com/guide/non-functional-requirements-examples> [Accessed 1 April 2025].

Ali, E., 2024. *Service-Level Agreement (SLA).* [online] Linkedin.com. Available at: <https://www.linkedin.com/pulse/understanding-service-level-agreements-slas-cloud-computing-ali-ah7bf> [Accessed 1 April 2025].

altexsoft, 2022. *Non-functional Requirements: Examples, Types, How to Approach*. [online] AltexSoft. Available at:< https://www.altexsoft.com/blog/non-functional-requirements/> [Accessed 1 April 2025].

AWS, 2023. *What is Multi-Factor Authentication (MFA)? - Cloud Security Beginner's Guide - AWS*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/what-is/mfa/> [Accessed 31 March 2025].

AWS, 2025. *What is Database Sharding? - Database Sharding Explained - AWS*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/what-is/database-sharding/> [Accessed 1 April 2025].

Barrett, K., 2023. *Guide to Understanding Non-Functional Requirements*. [online] QAT Global. Available at: <https://qat.com/guide-understanding-non-functional-requirements/> [Accessed 1 April 2025].

ByteByteGo, 2023. *Top 5 Most Used Architecture Patterns*. [online] YouTube. Available at: <https://www.youtube.com/watch?v=f6zXyq4VPP8> [Accessed 3 April 2025].

ByteByteGo, 2024a. *Software Architecture Patterns*. [online] Bytebytego.com. Available at: <https://blog.bytebytego.com/p/software-architecture-patterns> [Accessed 8 April 2025].

ByteByteGo, 2024b. *Stateless Architecture: The Key to Building Scalable and Resilient Systems*. [online] Bytebytego.com. Available at: <https://blog.bytebytego.com/p/stateless-architecture-the-key-to> [Accessed 1 April 2025].

Canva, 2016. *Factory Method Design*. [online] Canva. Available at: <https://www.canva.com/design/DAGkDaWq80s/ZZBacX-3RvFKjpbiHumdkw/edit?utm_content=DAGkDaWq80s&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton> [Accessed 3 April 2025].

Cato Networks, 2024. *6 Network Security Protocols You Should Know*. [online] Cato Networks. Available at: <https://www.catonetworks.com/network-security/network-security-protocols/> [Accessed 1 April 2025].

Cheth Virajini, 2023. *Design Patterns-Object-Oriented software design - Cheth Virajini - Medium*. [online] Medium. Available at: <https://medium.com/@byte.talking/design-patterns-object-oriented-software-design-0f454d630c9b> [Accessed 3 April 2025].

Creative Design Assets, n.d. *Scalability*. Available at: <https://i.pinimg.com/736x/fb/33/b1/fb33b16a849eb4d2e3ead2bba3f7e30d.jpg> [Accessed 1 April 2025].

Design Gurus, 2024. *Is performance a non-functional requirement?* [online] Tech Interview Preparation – System Design, Coding & Behavioral Courses | Design Gurus. Available at: <https://www.designgurus.io/answers/detail/is-performance-a-non-functional-requirement> [Accessed 2 April 2025].

Devduni, B., 2022. *Factory Method Design Pattern*. [online] Medium. Available at: <https://medium.com/@bhagya.devduni98/factory-method-design-pattern-59e3c0243230> [Accessed 3 April 2025].

Dilmegani, C., 2025. *Role-based access control (RBAC) in 2025.* Available at: <https://research.aimultiple.com/rbac/> [Accessed 1 April 2025].

Dinidu Sachintha, 2025. *Understanding the Facade Design Pattern: Simplifying Complex Systems*. [online] Medium. Available at: <https://medium.com/@dinidusachintha/understanding-the-facade-design-pattern-simplifying-complex-systems-a51416192c7b> [Accessed 2 April 2025].

Doshi, K., 2023. *Types of Software Testing: Learn with Examples*. [online] BrowserStack. Available at: <https://www.browserstack.com/guide/types-of-testing> [Accessed 2 April 2025].

Dwyer, J., 2023. *Ultimate Guide On Deployment Automation | Zeet.co*. [online] zeet.co. Available at: <https://zeet.co/blog/deployment-automation> [Accessed 2 April 2025].

EdPrice-MSFT, 2023. *Microservice architecture style - Azure Architecture Center*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>[Accessed 2 April 2025].

Emerich, A., 2023. *Database caching: Overview, types, strategies and their benefits.* [online] Prisma's Data Guide. Available at: <https://www.prisma.io/dataguide/managing-databases/introduction-database-caching> [Accessed 2 April 2025].

Faruqui, Z., 2024. *Automated Deployment Tools and Cloud Efficiency*. [online] Withcoherence.com. Available at: <https://www.withcoherence.com/articles/automated-deployment-tools-and-cloud-efficiency> [Accessed 31 March 2025].

Faster Capital, 2024. *1.How can I optimize images on my website to improve SEO and enhance user experience?* Available at: <https://fastercapital.com/keyword/lazy-loading-images.html> [Accessed 1 April 2025].

Foster, S., 2020. *What Are Secure Coding Standards? Security Standards Overview*. [online] Perforce Software. Available at: <https://www.perforce.com/blog/qac/secure-coding-standards> [Accessed 31 March 2025].

Full Stack Developer, 2024. *Fail Fast and Fail-Safe Design Principles — With Java Code Examples*. [online] Medium. Available at: <https://medium.com/@ByteCodeBlogger/fail-fast-and-fail-safe-design-principles-with-java-code-examples-420f3af9fd24> [Accessed 1 April 2025].

Ganesh, R., 2024. *Database Caching strategies explained! - R. Ganesh - Medium*. [online] Medium. Available at: <https://medium.com/@rganesh0203/database-caching-strategies-explained-6e02774cb674 > [Accessed 1 April 2025].

Hornay, R., 2021. *Deployment Automation*. Available at: <https://www.bmc.com/blogs/deployment-automation-benefits/> [Accessed 1 April 2025].

Imperva, 2019. *What is Minification | Why minify JS, HTML, CSS files | CDN Guide | Imperva*. [online] Learning Center. Available at: <https://www.imperva.com/learn/performance/minification/> [Accessed 2 April 2025].

IP Switch, 2025. *System Configuration - Firewall Configuration*. [online] Ipswitch.com. Available at: <https://docs.ipswitch.com/MOVEit/DMZ%207.5/online%20guide/MOVEitDMZ_SystemConfiguration_FirewallConfiguration.htm> [Accessed 1 April 2025].

Jurran, M., 2024. *Basics of Design Patterns: Template Method | The Software Design Handbook*. [online] Medium. Available at: <https://softwaredesignhandbook.com/basics-of-design-patterns-software-engineering-for-beginners-template-method-edc6f00c8c05> [Accessed 3 April 2025].

Khatri, H.K., 2025. *Mastering Data Handling in Flutter: Lazy Loading vs Pagination*. [online] Medium. Available at: <https://mailharshkhatri.medium.com/mastering-data-handling-in-flutter-lazy-loading-vs-pagination-6b14888d3c85> [Accessed 1 April 2025].

Kirvan, P., 2025. *What is security by design? - Definition from WhatIs.com*. [online] WhatIs.com. Available at: <https://www.techtarget.com/whatis/definition/security-by-design> [Accessed 31 March 2025].

Komninos, A., 2016. *An Introduction to Usability*. [online] The Interaction Design Foundation. Available at: <https://www.interaction-design.org/literature/article/an-introduction-to-usability?srsltid=AfmBOooVtnFYygdETkrWOgFVzC_Hv_UpxwQtgaYPQ6U0xOWPbEKCLjiO> [Accessed 1 April 2025].

Kumar, A., 2023. *Facade design pattern: - Atul Kumar - Medium*. [online] Medium. Available at: <https://medium.com/@kumar.atul.2122/facade-design-pattern-f7d3aba2560b> [Accessed 3 April 2025].

LearnCsDesign, 2022. *Learn the Facade Design Pattern - LEARNCSDESIGN*. [online] LEARNCSDESIGN - An easy way to learn CS concepts. Available at: <https://www.learncsdesign.com/learn-the-facade-design-pattern/#google_vignette> [Accessed 3 April 2025].

Lindemulder, G. and Kosinski, M., 2024. *What is role-based access control (RBAC)?* [online] IBM. Available at: <https://www.ibm.com/think/topics/rbac> [Accessed 1 April 2025].

Oracle, 2021. *What is a relational database?* [online] www.oracle.com. Available at: <https://www.oracle.com/za/database/what-is-a-relational-database/> [Accessed 1 April 2025].

Paredes, R., 2023. *Performance Benchmarking: A Quick Guide*. [online] SafetyCulture. Available at: <https://safetyculture.com/topics/benchmarking/performance-benchmarking/> [Accessed 2 April 2025].

Patel, M., 2024. *Leveraging Cloud Services for Efficient Web Development*. [online] CloudThat Resources. Available at: <https://www.cloudthat.com/resources/blog/leveraging-cloud-services-for-efficient-web-development> [Accessed 1 April 2025].

POPIA, 2021. *Protection of Personal Information Act (POPI Act)*. [online] POPIA. Available at: <https://popia.co.za/> [Accessed 31 March 2025].

RedHat, 2022. *What is container orchestration?* [online] www.redhat.com. Available at: <https://www.redhat.com/en/topics/containers/what-is-container-orchestration> [Accessed 1 April 2025].

Refactoring Guru, 2014. *What's a Design pattern?* [online] Refactoring.guru. Available at: <https://refactoring.guru/design-patterns/what-is-pattern> [Accessed 2 April 2025].

RescoNext, 2025. *Simple guide to failure metrics (MTBF vs. MTTR vs. MTTF)*. [online] Resco.net. Available at: <https://www.resco.net/learning/failure-metrics/> [Accessed 1 April 2025].

Saifi, N., 2023. *Non-Functional Requirements: Definition and Examples*. [online] Glossary. Available at: <https://chisellabs.com/glossary/what-is-non-functional-requirements/> [Accessed 1 April 2025].

Scale Computing, 2023. *IT Infrastructure Components*. [online] Scale Computing. Available at: <https://www.scalecomputing.com/resources/it-infrastructure-components> [Accessed 2 April 2025].

Shah, E., 2024. *Exploring the Factory Method Design Pattern - Eshika Shah - Medium*. [online] Medium. Available at: <https://medium.com/@eshikashah2001/exploring-the-factory-method-design-pattern-4d270b6ff935> [Accessed 3 April 2025].

Singla, L., 2023. *What are Microservices?* Available at: <https://www.netsolutions.com/hub/mach-architecture/microservices/> [Accessed 1 April 2025].

Source Making, 2025. *Design Patterns and Refactoring*. [online] sourcemaking.com. Available at: <https://sourcemaking.com/design_patterns/template_method> [Accessed 1 April 2025].

Swimm Team, 2024. *Swimm*. [online] Swimm. Available at: <https://swimm.io/learn/microservices/microservices-architecture-benefits-and-how-to-adopt> [Accessed 1 April 2025].

TechGiga, 2020. *Cloud computing: Everything you need to understand about the cloud.* Available at: <https://www.techgiga.net/cloud-computing/>.

Velu Dhanesh, 2018. *Let's Begin With 3-Tire Architecture - Velu Dhanesh - Medium*. [online] Medium. Available at: <https://medium.com/@velusamyvenkatraman/lets-begin-with-3-tire-architecture-39c1b3f9fa51> [Accessed 3 April 2025].

Verpex, 2023. *Lazy Loading vs Pagination*. [online] Verpex. Available at: <https://verpex.com/blog/website-tips/lazy-loading-vs-pagination> [Accessed 1 April 2025].

Watson, M., 2020. *10 Powerful Software Scalability Strategies for Unstoppable Growth*. [online] Full Scale. Available at: <https://fullscale.io/blog/software-scalability/> [Accessed 31 March 2025].

Wright, G., 2023. *What is uptime and downtime? - Definition from WhatIs.com*. [online] WhatIs.com. Available at: <https://www.techtarget.com/whatis/definition/uptime-and-downtime> [Accessed 2 April 2025].

Yalanska, M., 2020. *User Experience: Insights Into Consistency in Design*. [online] Tubik Blog: Articles About Design. Available at: <https://blog.tubikstudio.com/design-consistency/> [Accessed 2 April 2025].