Software Architecture Documentation

<E-Farm>

Software Requirements Specifications Project

Team Members:

MD ASIF MAHMUD | MUH2025004M

RUBYA RASHED | MUH2025014M

IMTIAZ CHOWDHURY | MUH2025027M

MAHAMUDUL HASAN TALUKDER | MUH2025028M

MEHEDI HASAN | MUH2025032M

Faculty Coach:

Dipok Chandra Das

Assistant Professor

Institute of Information Technology

Noakhali Science and Technology University

Contents

Table

Figure

Revision History

Date	Version	Description	Author
13/02/2024	v1.0	Initial revision	Md Asif Mahmud Rubya Rashed Imtiaz Chowdhury Mahamudul Hasan Talukder Mehedi Hasan

1. Introduction

In today's ever-evolving world of agriculture, technology is playing a crucial role in making farming more productive, efficient, and sustainable. Acknowledging this need for progress, the E-farm project emerges as a digital platform poised to transform the way farmers and consumers interact in the agricultural world.

1.1 Purpose

The main goal of this document is to explain the architectural framework of the E-farm project. It aims to provide a clear understanding of how the software is designed, including its main principles, parts, and functions. By doing so, this document helps stakeholders, developers, and contributors involved in building and maintaining E-farm to stay on the same page.

1.2 Scope

The software architecture outlined herein encompasses a multifaceted approach towards addressing the diverse needs of farmers and consumers engaged in agricultural activities. From facilitating direct transactions between producers and buyers to providing access to equipment rental, maintenance services, and expert consultancy, the E-farm platform aspires to streamline agricultural operations while fostering sustainable practices and economic growth.

1.3 Definitions, Acronyms and Abbreviations

To ensure clarity and comprehension throughout this document, the following definitions, acronyms, and abbreviations are provided:

E-farm: The digital platform designed to revolutionize agricultural practices.

Module: A discrete functional unit within the software architecture.

UI: User Interface.

API: Application Programming Interface.

1.4 References

I. 2011 Ninth Working IEEE/IFIP *Conference on Software Architecture*

II. IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998.

III. Software Architecture: *Foundations, theory and practice. Richard N. Taylor, Nenad Medvidovic, Eric Dashofy. Wiley, 2010*

IV. EBW Software Design: *From Programming to Architecture. Eric Braude. Wiley, 2004*

V. FRH Pattern Oriented Software Architecture: *A System of Patterns. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. Wiley, 1995*

VI. Visual Paradigm

1.5 Overview

As we embark on the journey to modernize agriculture through digital innovation, E-farm's software architecture demonstrates our commitment to using technology for the benefit of farmers and the environment. This introduction sets the stage for a deeper dive into the details of how E-farm works and what it offers.

2. Architecturally Significant Requirements

Performance: Requirements related to system response times, throughput, and scalability. These requirements may dictate the need for specific architectural patterns, caching strategies, or distributed systems.

Security: The platform must be secure to protect sensitive information such as user data, financial information, and transaction details. This includes data encryption, secure payment gateways, and secure user authentication.

Scalability: The platform must be designed to handle many users, transactions, and data. It should scale easily and efficiently as the user base and demand grow.

Maintainability: Requirements related to the ease of maintaining and updating the system over time. This may involve modular design, clean code practices, and documentation standards.

Interoperability: The platform needs to integrate with various third-party services such as payment gateways, logistics providers, and other relevant services. This requires an API-based architecture that can easily integrate with other systems.

Availability: The platform needs to be highly available and reliable to ensure users can always access the platform and its services. This includes measures such as redundancy, load balancing, and disaster recovery.

Data Management: The platform needs to be designed to manage and store large amounts of data related to users, products, transactions, and other relevant information. This includes measures such as database design, data security, and data backup.

User Experience: The platform needs to provide a seamless and intuitive user experience for all stakeholders, including farmers, consumers, agro-solution providers, agricultural equipment providers, logistics providers, and financial institutions. This includes features such as easy navigation, responsive design, and personalized recommendations.

3. Quality Attributes

Short Description of System Quality Attributes in E-Farm:

Availability: Ensuring the platform is consistently accessible and operational for both farmers and consumers, especially during critical times such as peak farming seasons or high consumer demand.

Reliability: Providing consistent and dependable services to farmers and consumers, ensuring that transactions, equipment rentals, and consultancy services work reliably without failures.

Usability: Designing an intuitive and user-friendly interface for both farmers and consumers to easily navigate the platform, making transactions, rentals, and consultations straightforward.

Performance: Optimizing the speed and efficiency of the platform to handle transactions, equipment bookings, and consultations swiftly and effectively, especially during high-traffic periods.

Security: Ensuring robust security measures to protect the sensitive data of both farmers and consumers, especially in financial transactions and personal information.

Scalability: The platform's ability to accommodate an increasing number of users, farmers, and consumers without compromising performance or service quality.

Maintainability: Ease of maintaining and updating the platform, including system updates, bug fixes, and introducing new features without disrupting ongoing services.

Interoperability: The ability of the platform to seamlessly interact with different agricultural tools, equipment, and systems used by farmers, ensuring compatibility and integration.

3.1 Quality Attributes Scenarios

Scenario for System Quality Attributes in E-Farm:

Table 1 Availability Scenario: High Traffic Period

Scenario	A surge in user activity during the planting season
Source of Stimulus	Surge in user activity
Stimulus	Increased user activity during the planting season
Artifact	Server processors, communication channels
Environment	Normal operation
Response	Auto-scaling of server resources, load balancing
Measure	Uptime maintained at 99.9%, with minimal impact on transaction speed

Table 2 Scenario: Scheduled Maintenance

Scenario	The E-Farm platform requires scheduled maintenance to apply updates and perform system optimizations
Source of Stimulus	System maintenance schedule
Stimulus	Initiation of scheduled maintenance
Artifact	Maintenance notification system, maintenance schedule database.
Environment	Planned maintenance window

Response	Temporary suspension of services with a clear notification to users, redirection to a maintenance page.
Measure	Uptime maintained at 99.9% over the long term, with no unexpected downtime

Table 3 Scenario: Network Outage Recovery

Scenario	A network outage occurs due to an unexpected event such as a power outage or infrastructure failure.
Source of Stimulus	Network outage event
Stimulus	Loss of connectivity to the E-Farm platform
Artifact	Network monitoring system, failover mechanisms.
Environment	Abnormal operation due to network outage.
Response	Automatic failover to redundant network connections or backup servers located in different geographical regions
Measure	Minimal downtime experienced by users

Table 4 Reliability Scenario: Equipment Rental

Scenario	Farmer requests equipment rental
Source of Stimulus	Farmer requests equipment rental
Stimulus	Equipment rental request
Artifact	Rental booking system, equipment availability database
Environment	Normal operation
Response	Immediate confirmation of booking, accurate equipment availability information
Measure	Zero booking errors, timely and accurate equipment delivery

Table 5 Scenario: Agro Solution Consultancy Appointment

Scenario	A farmer schedules an appointment for agro solution consultancy through the E-Farm platform
Source of Stimulus	Farmer schedules an appointment for agro solution consultancy.
Stimulus	Appointment scheduling request
Artifact	Appointment booking system, consultant availability database.
Environment	Normal operation
Response	Immediate confirmation of appointment booking, accurate consultant availability information
Measure	Zero scheduling conflicts, timely and accurate consultancy sessions.

Table 6 Scenario 2: Product Delivery Tracking

Scenario	A consumer places an order for agricultural products through the E-Farm platform
Source of Stimulus	Consumer places an order and wants to track delivery
Stimulus	Delivery tracking request
Artifact	Delivery tracking system, order status database
Environment	Normal operation
Response	Real-time updates on the delivery status, including estimated time of arrival and any delays.
Measure	Accurate and up-to-date delivery tracking information, timely delivery within the specified timeframe.

Table 7 Usability Scenario: User Interface Navigation

Scenario	New consumers exploring the platform for produce purchase
Source of Stimulus	New consumers exploring the platform for produce purchase
Stimulus	Consumer exploring platform
Artifact	User interface, navigation menus
Environment	Normal operation
Response	Clear and intuitive interface, easily navigable categories for produce selection
Measure	Low learning curve, successful purchase within 5 minutes

Table 8 Scenario: Product Search and Filter

Scenario	A returning consumer wants to quickly find specific products based on their preferences
Source of Stimulus	Consumer searching for specific products
Stimulus	Search and filter actions performed by the consumer
Artifact	Product search and filtering features in the user interface.
Environment	Normal operation
Response	Clear presentation of search results with detailed product descriptions and images.
Measure	Average search and selection time is less than 2 minutes. Curate and up-to-date delivery tracking information, timely delivery within the specified timeframe.

Table 9 Scenario: Account Registration and Onboarding

Scenario	A new farmer wants to create an account and onboard onto the E-Farm platform to start selling their products.
Source of Stimulus	New farmer interested in joining the platform.
Stimulus	Account registration and onboarding process.
Artifact	User registration form, onboarding tutorials or guidance
Environment	Normal operation
Response	User-friendly registration form with clear instructions and minimal required fields Lear presentation of search results with detailed product descriptions and images.
Measure	Farmers successfully complete the onboarding process and list their products within 10 minutes.

Table 10 Performance Scenario: Consultation Service

Scenario	High concurrent consultation requests
Source of Stimulus	High concurrent consultation requests
Stimulus	Concurrent consultation requests
Artifact	Consultation scheduling system, server response time
Environment	Normal operation
Response	Swift scheduling and connection with consultants, minimal wait time
Measure	Average response time below 10 seconds, no service disruptions

Table 11 Scenario: Marketplace Product Listing

Scenario	Multiple farmers simultaneously list their products on the E-Farm marketplace
Source of Stimulus	Simultaneous product listing by multiple farmers
Stimulus	Concurrent product listing requests
Artifact	Product listing system, server resources
Environment	Normal operation
Response	Efficient handling of concurrent product listing requests with minimal delay.
Measure	Average response time for product listing requests remains below 5 seconds

Table 12 Scenario: Order Processing

Scenario	A surge in orders occurs during peak shopping hours on the E-Farm platform

Source of Stimulus	Increase in orders during peak shopping hours
Stimulus	Concurrent order placement requests
Artifact	Order processing system, server resources
Environment	Normal operation
Response	Prompt processing of orders to prevent delays in order confirmation and fulfillment
Measure	Average order processing time remains below 10 seconds

Table 13 Security Scenario: Financial Transaction

Scenario	Consumer purchasing produce using a payment gateway
Source of Stimulus	Consumer purchasing produce using a payment gateway
Stimulus	Purchase transaction via payment gateway
Artifact	Consultation scheduling system, server response time
Environment	Normal operation
Response	Secure payment processing, encryption of sensitive data
Measure	No reported security breaches, encrypted data transmission maintained

Table 14 Scenario: Farmer accessing financial information

Source of Stimulus	Farmer accessing financial information for transaction or account management.
Stimulus	Farmer accessing financial records or initiating a transaction.
Artifact	Financial database, user authentication system.
Environment	Normal operation.
Response	Secure authentication process, access to authorized financial data.
Measure	No unauthorized access reported, secure login process maintained.

Table 15 Scenario: Third-party attempting unauthorized access

Source of Stimulus	Unauthorized third-party attempting to access financial data.
Stimulus	Unauthorized login attempts, suspicious activity detected.
Artifact	Firewall, intrusion detection system, authentication logs.
Environment	Normal operation.
Response	System detects and blocks unauthorized access attempts, alerts system administrators.
Measure	No successful unauthorized access reported, intrusion attempts detected and mitigated.

Table 16 Scalability Scenario: Rapid User Growth

Scenario	Increased user registrations

Source of Stimulus	Increased user registrations
Stimulus	Increase in user registrations
Artifact	Server infrastructure, user database
Environment	Normal operation
Response	Auto-scaling of server resources, seamless onboarding of new users
Measure	Platform performance maintains stability, and no degradation despite an increased user base

Table 17 Scenario: High Traffic Event

Source of Stimulus	High-profile marketing campaign or promotional event leading to a surge in website traffic.
Stimulus	Sudden influx of users accessing the platform simultaneously.
Artifact	Server infrastructure, load balancer, website traffic monitoring tools.
Environment	Normal operation.
Response	Automatic scaling of server resources to accommodate increased traffic, temporary increase in bandwidth allocation.
Measure	Platform maintains responsiveness and stability during peak traffic, no downtime reported despite the surge in users.

Table 18 Scenario: Geographic Expansion

Source of Stimulus	Expansion of the platform's services to new geographical regions.
Stimulus	Increase in user registrations and usage from new geographic areas.
Artifact	Server infrastructure, user database, localization features.
Environment	Normal operation.
Response	Scaling up server resources to support the growing user base in new regions, implementing localized content and services.
Measure	Platform remains accessible and responsive in new regions, no performance degradation observed despite the expansion.

Table 19 Maintainability Scenario: System Update

Scenario	Introduction of a new feature set for equipment monitoring
Source of Stimulus	Introduction of a new feature set for equipment monitoring
Stimulus	New feature introduction
Artifact	Software architecture, update deployment process
Environment	Normal operation
Response	Seamless deployment of updates, minimal service disruption
Measure	No downtime during the update, and successful feature integration

Table 20 Scenario: Security Patch Deployment

Source of Stimulus	Identification of critical security vulnerabilities in the system.
Stimulus	Release of security patches to address identified vulnerabilities.
Artifact	Software architecture, update deployment process, security monitoring

	tools.
Environment	Normal operation.
Response	Prompt deployment of security patches to all affected components of the system, including servers, databases, and application code.
Measure	No security breaches reported after the deployment of patches, system remains secure and resilient against potential threats.

Table 21 Scenario: Technology Stack Upgrade

Source of Stimulus	Outdated technology stack components posing compatibility and performance risks.
Stimulus	Decision to upgrade underlying technologies such as programming languages, frameworks, or database systems.
Artifact	Software architecture, technology upgrade roadmap, compatibility testing environment.
Environment	Normal operation.
Response	Methodical planning and execution of technology stack upgrades, including compatibility testing with existing system components.
Measure	Successful implementation of upgraded technologies without service disruptions, improved system performance and stability following the upgrade.

Table 22 Interoperability Scenario: Integration with Farming Equipment

Scenario	Farmer attempts to link personal monitoring equipment to the platform
Source of Stimulus	Farmer attempts to link personal monitoring equipment to the platform
Stimulus	Integration of personal farming equipment
Artifact	API, equipment compatibility
Environment	Normal operation
Response	Successful integration, and data synchronization between equipment and platform
Measure	No data discrepancies, real-time and e equipment data displayed accurately

Table 23 Scenario: Integration with Third-Party Weather Sensors

Source of Stimulus	Farmers seeking to incorporate data from third-party weather sensors into the platform.
Stimulus	Request for integration of weather sensor data with the farming platform.
Artifact	API for weather sensor integration, compatibility testing environment.
Environment	Normal operation.
Response	Development and implementation of an API to integrate weather sensor data with the platform. This includes ensuring compatibility with various weather sensor models and protocols.
Measure	Successful integration of weather sensor data into the platform, with accurate synchronization and display of real-time weather information. No discrepancies between weather data obtained from sensors and other sources.

Table 24 Scenario: Integration with Agricultural Machinery Telemetry Systems

Source of Stimulus	Farmers aiming to connect their agricultural machinery telemetry systems to the platform.
Stimulus	Request for integration of telemetry data from agricultural machinery.
Artifact	API for machinery telemetry integration, compatibility testing environment.
Environment	Normal operation.
Response	Development and deployment of an API to facilitate the integration of telemetry data from various agricultural machinery brands and models. This involves ensuring compatibility and interoperability with different telemetry protocols and data formats.
Measure	Successful integration of telemetry data from agricultural machinery into the platform, with accurate synchronization and display of real-time equipment performance metrics. No discrepancies between telemetry data and manual equipment inspections.

3.2 Utility Trees

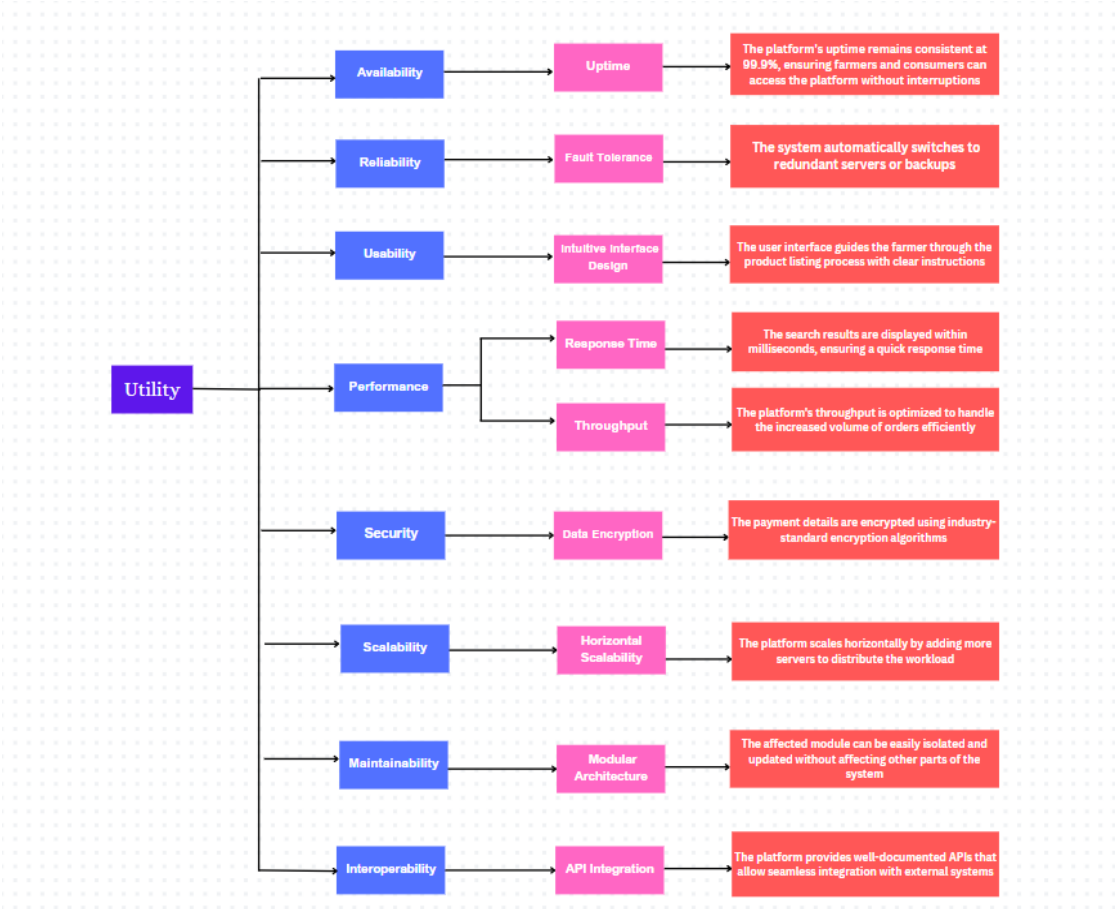

Figure 1 Utility Trees

3.3 Design Tactics

3.3.1 Tactics for Availability

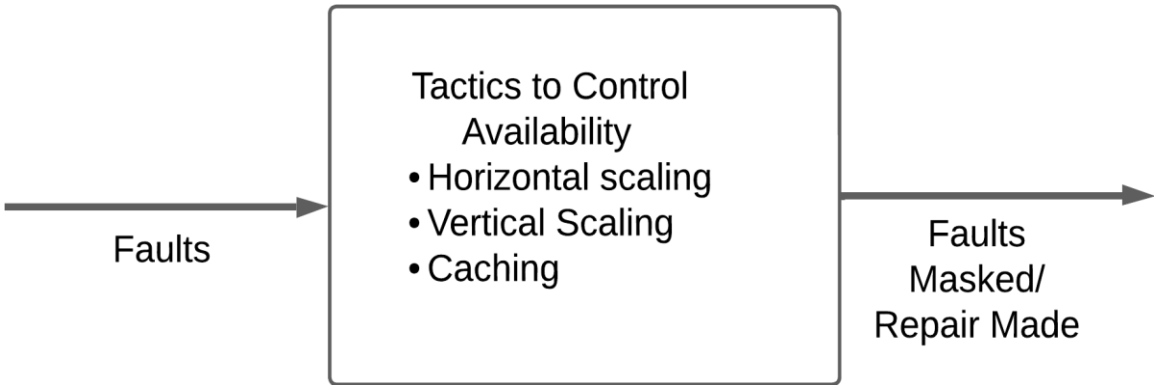

Figure 2 Tactics for Availability

- **Horizontal scaling:** This involves adding more machines to distribute the workload, improving uptime, and handling more requests.
- **Vertical scaling:** This involves upgrading a single machine's resources, like CPU or memory, to handle a heavier load.
- **Caching:** This involves storing frequently accessed data in a temporary location for faster retrieval, reducing load on the main system.
- **Masked repair:** This involves fixing an error without taking the system offline, minimizing downtime and disruption.

3.3.2 Tactics for Usability

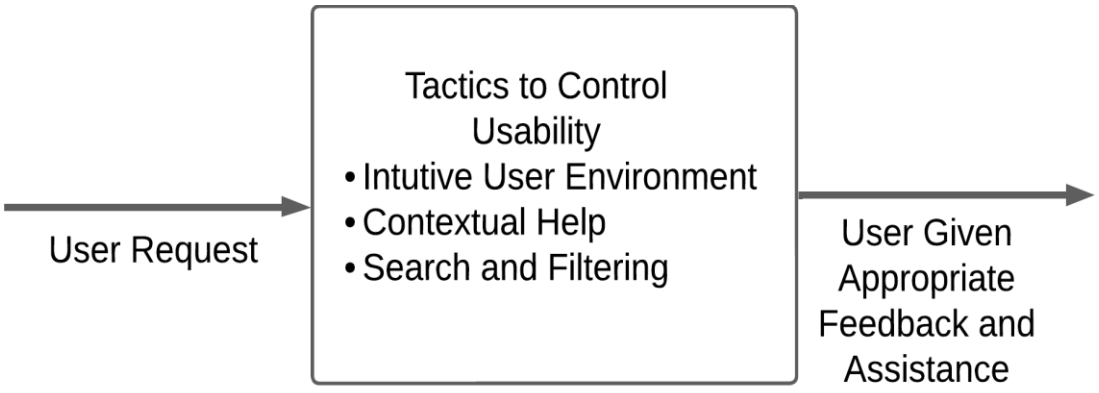

Figure 3 Tactics for Usability

- **Intuitive User Environment:** This refers to designing an interface that is easy to understand and use, without the need for a lot of training or explanation. This can include using clear and concise language, consistent layout, and familiar icons and elements.

- **Contextual Help:** This means providing help and support that is relevant to the specific task that the user is trying to complete. This can be done through tooltips, pop-up windows, or on-screen instructions.
- **Search and Filtering:** This allows users to quickly find the information they are looking for by providing them with ways to search and filter through content. This can be especially helpful for users who are working with large amounts of data.

3.3.3 Tactics for Performance

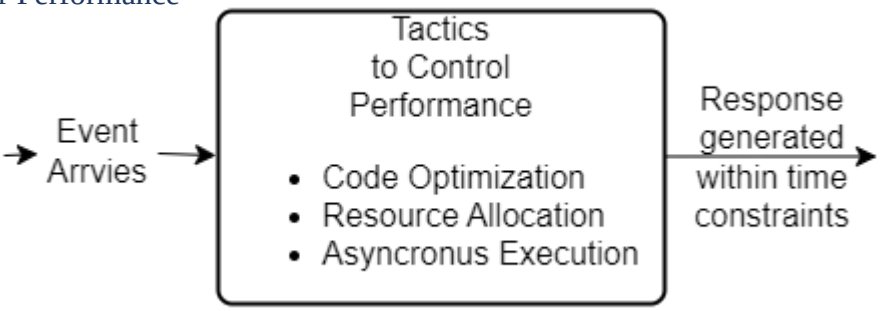

Figure 4 Tactics for Performance

- **Code optimization:** This involves making changes to code to make it run more efficiently. This can include things like reducing the number of calculations that are performed or using more efficient data structures.
- **Resource allocation:** This involves allocating resources, such as memory and CPU time, to different tasks in a way that optimizes performance.
- **Asynchronous execution:** This involves executing tasks concurrently, rather than one after the other. This can improve performance by allowing tasks to overlap with each other.

3.3.4 Tactics for Security

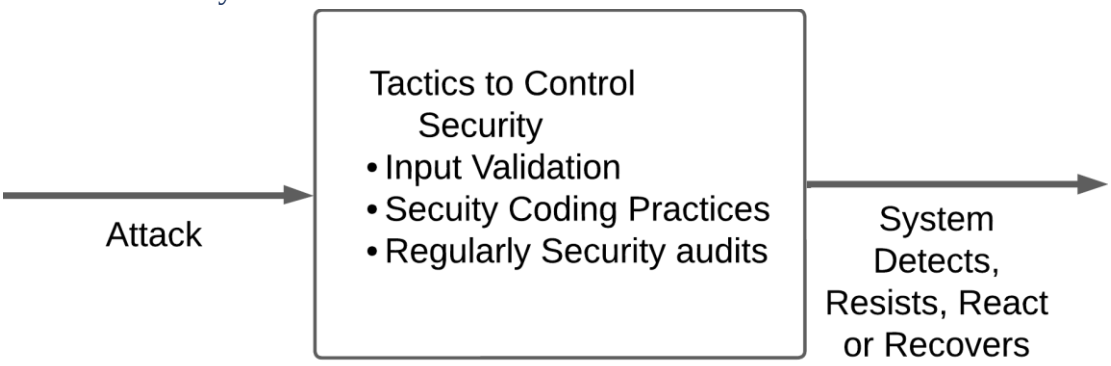

Figure 5 Tactics for Security

- **Input Validation:** This involves checking that all data inputs from users or other sources are valid and safe before being processed by the system. This helps to prevent errors and security vulnerabilities.
- **Security Coding Practices:** This refers to following secure coding practices when developing software to avoid introducing security weaknesses. This includes things like avoiding buffer overflows, SQL injection, and other common coding vulnerabilities.
- **Regular Security Audits:** This involves regularly conducting security audits to identify and address any potential security risks in the system. This can help to ensure that the system is protected from the latest threats.

3.3.5 Tactics for Interoperability

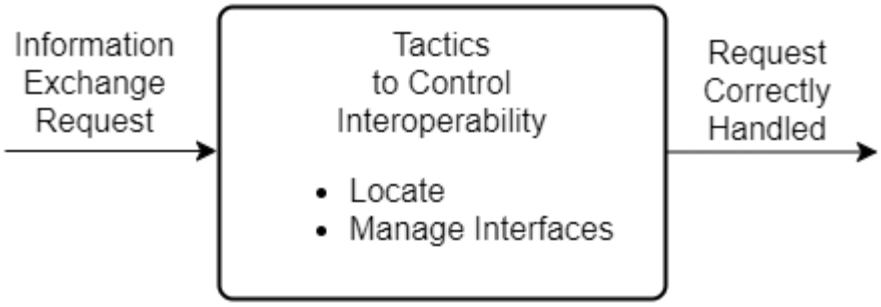

Figure 6 Tactics for Interoperability

- **Locate:** This tactic involves identifying the specific interfaces where data will be exchanged between the systems.
- **Control:** This tactic involves establishing rules and procedures for how data will be exchanged between the systems, such as security measures and error handling.

3.3.6 Tactics for Modifiability

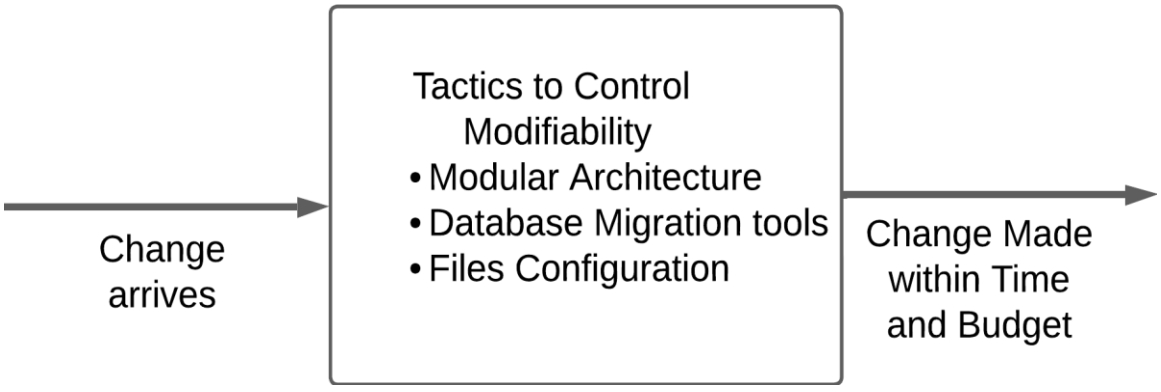

Figure 7 Tactics for Modifiability

Modular Architecture: This involves structuring the system into independent modules that can be easily changed without affecting other parts of the system.

Database Migration Tools: These tools automate the process of migrating data from one database to another, making it easier to change the database schema.

Files Configuration: This involves storing configuration data in external files, so that it can be easily changed without having to recompile the code.

4. Architectural Representation

4.1 Context Diagram

The context diagram for E-Farm

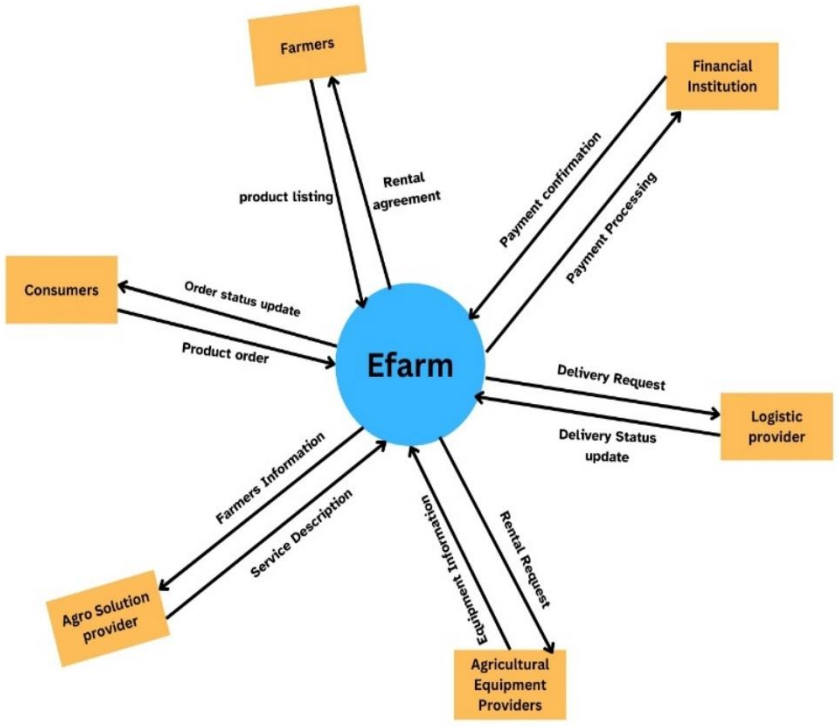

Figure 8 context diagram

4.2 Use-Case Diagram

What do we do?

E-Farm serves as a direct link between *farmers* and *consumers*, offering various services:

- Agri Marketplace
- Agricultural Equipment Rental and Maintenance
- Agro Solution and Consultancy

Use Case Diagram

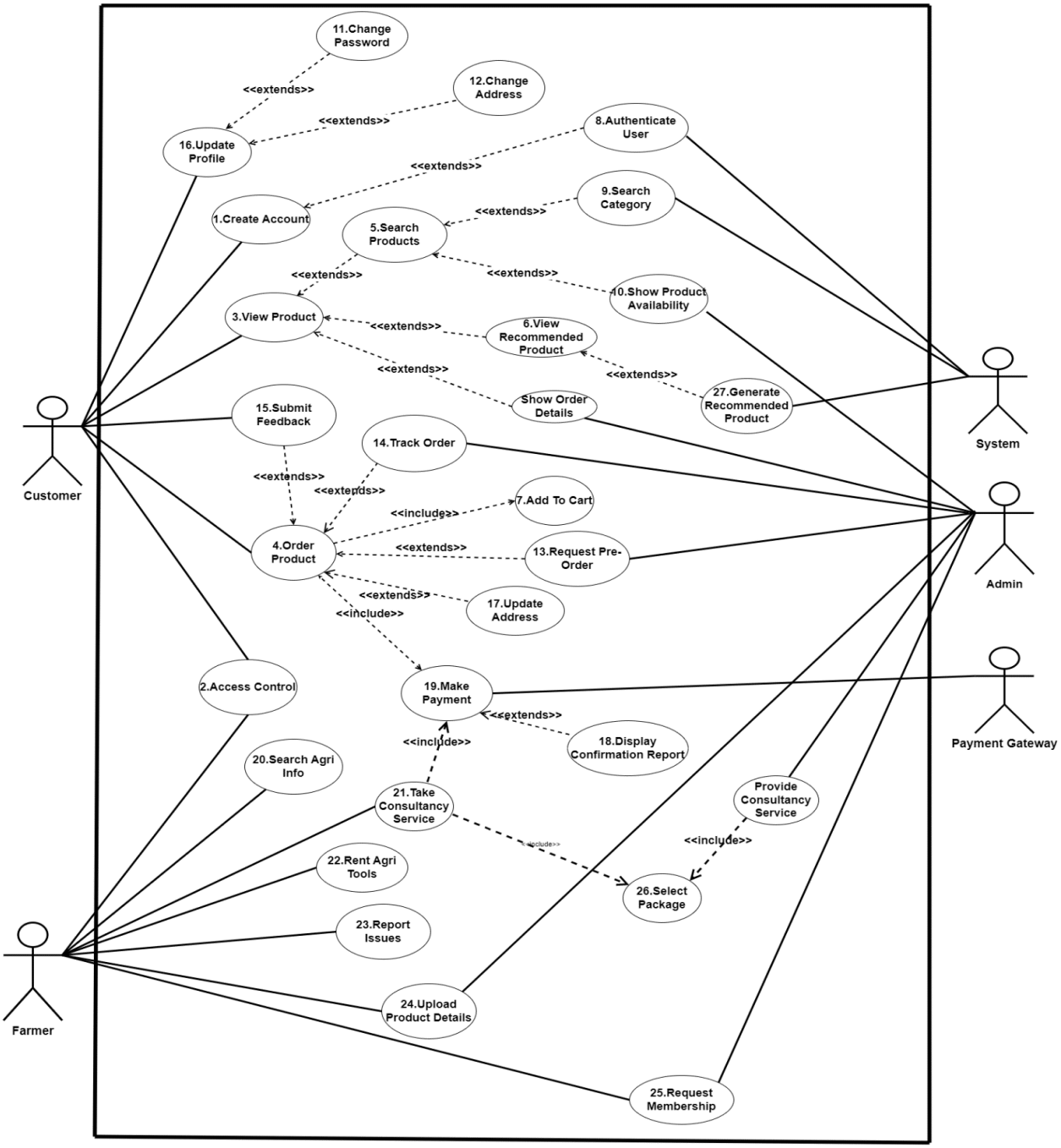

Figure 9 Use Case Diagram

1. The "**Create Account**" use case in E-Farm enables customers to register for an account. Customers input personal details, the system validates information, and upon success, creates the account and sends a confirmation email. Error messages prompt users to correct invalid or incomplete data, and if the username or email is already in use, users are prompted to choose alternatives.
2. The "**Access Control**" use case in E-Farm allows customers to log in to their accounts. Customers enter their credentials on the login page, and the system validates them. Upon success, customers gain access to their account information. Error messages prompt users to correct invalid credentials or contact support if the account is deactivated.

3. The "**View Product**" use case in E-Farm enables logged-in customers to browse available agricultural products. Customers are presented with a list that can be filtered by type or price. Upon selection, detailed information about the chosen product is displayed.

4. The "**Order Product**" use case in E-Farm enables logged-in customers to purchase agricultural products. Customers fill out a form with shipping and payment details, and upon validation, the order is confirmed. The system updates inventory and sends a confirmation email.

5. The "**Search Product**" use case in the platform allows customers to find desired agricultural products efficiently. Customers input search criteria, view matching products and add desired items to their cart.

6. The "**View Recommended Products**" use case enables customers to explore recommended agricultural products on the platform. Customers click the designated button, prompting the platform to display relevant recommendations based on their history. Success entails customers viewing and potentially adding recommended products to their cart.

7. The "**Add to Cart**" use case permits customers to add agricultural products to their shopping cart. Customers, with a valid account, select a product and click "Add to Cart." Success occurs when the chosen product is successfully added, and the cart updates accordingly. Customers then proceed to checkout or continue shopping.

8. The "**Authenticate Customer**" use case ensures secure access to customer accounts and personal information on the platform. Customers log in using credentials, which the platform

validates. Successful authentication grants access to account information and platform features. Alternatives include password resets for forgotten credentials.

9. The "**Search Category**" use case enables customers to explore products within specific categories in the E-Farm System. Customers select a category and view subcategories, followed by relevant products. They can filter products based on desired criteria. The system ensures customers efficiently navigate and explore products within chosen categories.

10. The "**Show Products Availability**" use case in the Agri Product Delivery System aims to display the availability of selected products to customers. Upon selecting a product, the system retrieves availability data from the database and promptly presents it to the customer.

11. The "**Select Delivery Types**" use case enables customers to choose delivery options for their agricultural product orders. Upon order placement, customers select a delivery type from available options. The system confirms and updates the order accordingly.

12. The "**Select Payments Types**" use case facilitates customers in choosing payment methods for their agricultural product orders. Upon order placement, customers select a payment type from available options. The system confirms and updates the order accordingly.

13. The "**Deliver Products**" use case involves delivering agricultural products to customers. After successful order placement and payment, the delivery system dispatches personnel to the customer's location. Upon delivery, the customer confirms receipt, and the system updates the status.
14. The "**Track Order**" use case enables customers to monitor the delivery status of their agricultural product orders. Customers select the "Track Order" option, and the system retrieves and displays the current delivery status. If issues arise, such as delays, the system updates the estimated delivery time and informs the customer.

15. The "**Submit Feedback**" use case allows customers to provide feedback on product quality, delivery service, or overall experience. Customers access the system, select "Submit Feedback," and enter their feedback.

16. The "**Update Profile**" use case enables customers to modify their profile information, including personal details and delivery preferences. Customers access the system, select "Update Profile," enter the updated information, and submit it.

17. The "**Update Address**" use case allows customers to modify their delivery address. Customers access the system, select "Update Address," input the new address information, and submit it. The system then updates the customer's address accordingly.

18. The "**Display Confirmation Report**" use case enables customers to view a confirmation report for their orders. Customers access the system, select "View Confirmation Report," and the system retrieves and displays the report promptly.

19. The "**Make Payment**" use case allows customers to complete payment for their orders securely and efficiently. Customers select the "Make Payment" option, and the system redirects them to a payment gateway. Upon entering payment information, the gateway processes the payment, and the system confirms it, finalizing the order.

20. The "**Search Agri Information**" use case enables farmers to find information relevant to their agricultural products. Farmers select the "Search Agri Information" option, and the system retrieves and displays the information related to their products promptly.

21. The "**Take Consultancy Service**" use case enables farmers to access consultancy services for their agriculture products. Farmers select the "Consultancy Service" option, and the system retrieves available services, connects them to a consultant, and provides necessary guidance.

22. The "**Rent Agri Tools**" use case enables farmers to rent agricultural tools for their farming activities. Farmers select the "Rent Agri Tools" option, choose the desired tool and rental period, and the system connects them to a rental service provider.

23. The "**Report Issues**" use case allows farmers to report encountered issues in the Agri Product Delivery System. Farmers select the "Report Issue" option, fill out a form detailing the issue, and submit it.

24. The "**Upload Product Details**" use case enables registered farmers to upload their product details onto the Agri Product Delivery System for direct sale to customers.

25. The "**Request Membership**" use case enables farmers to seek membership on the Agri Product Delivery System for selling their products. Farmers input their personal details, and upon approval by the system, they gain access to sell products on the platform.

26. The "**Select Package**" use case enables farmers to choose a package on the Agri Product Delivery System for selling their products at a discounted commission rate. Upon selection, the system applies the discounted commission rate to the farmer's product sales for the package duration.

27. The "**Generate Recommended Products"** use case allows customers to receive personalized product recommendations based on their preferences and past purchase history. The system retrieves customer data, generates a list of recommended products, and displays them for the customer to explore and potentially add to their cart.

4.3 Logical View

Focus: Functionality without implementation details.

Viewpoint Type: Functional Viewpoint.

Viewers: Developers, Architects, Business Analysts.

Viewpoint Style: *Class diagrams*, Package diagrams, Component diagrams.

Components: User interfaces, databases, marketplace, equipment services, agro solutions.

Considerations: Emphasize the functional aspects only, without delving into the technical implementation details.

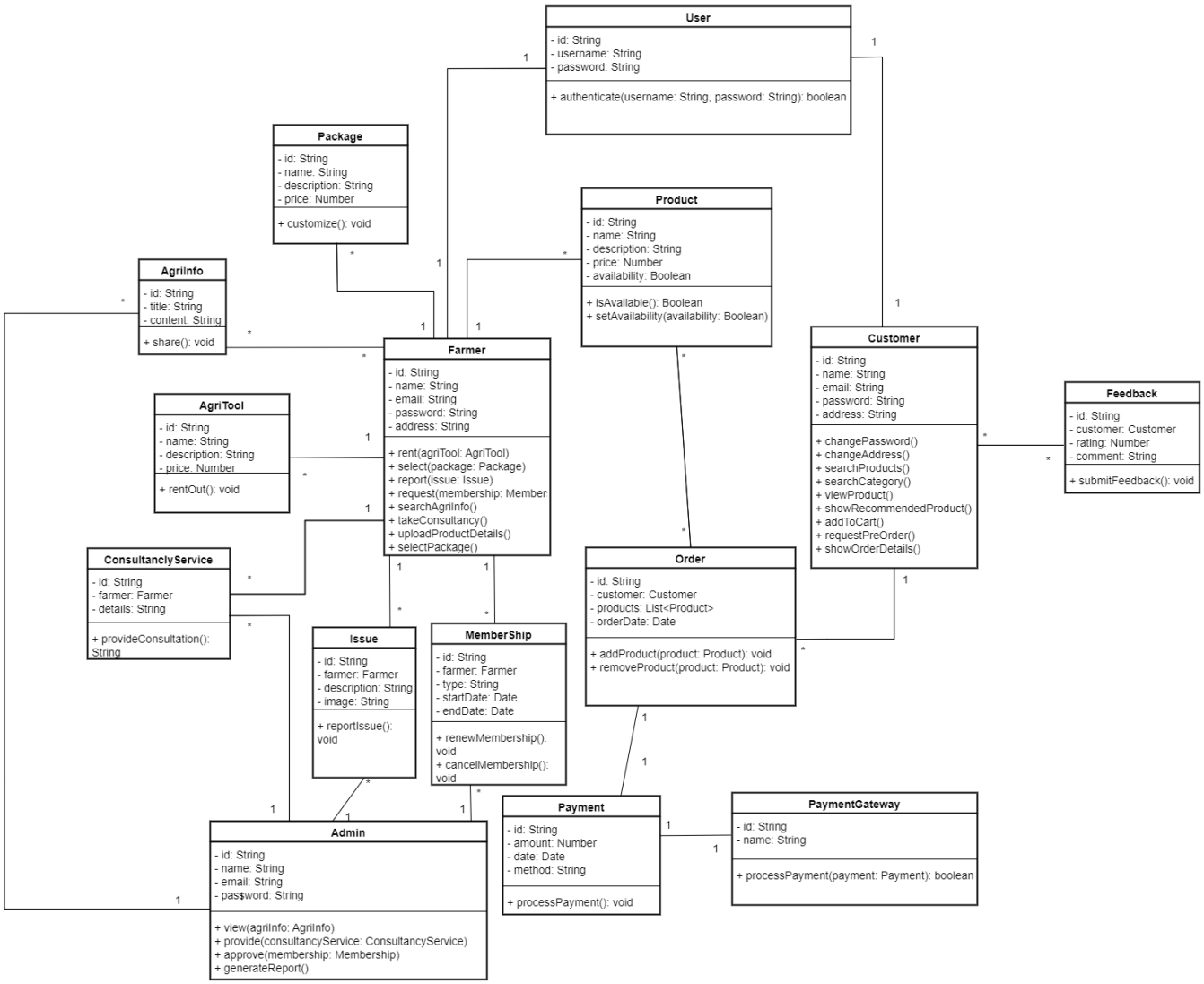

Figure 10 Class diagrams

4.4 Process View

Focus: System's dynamic behavior and interactions.

Viewpoint Type: Dynamic Viewpoint.

Viewers: Developers, System Analysts, QA Engineers.

Viewpoint Style: *Sequence diagrams*, Activity diagrams, State diagrams.

Components: Interaction flows (e.g., login, order placement, consultation service).

Considerations: Illustrate system behaviors and interactions among components/processes.

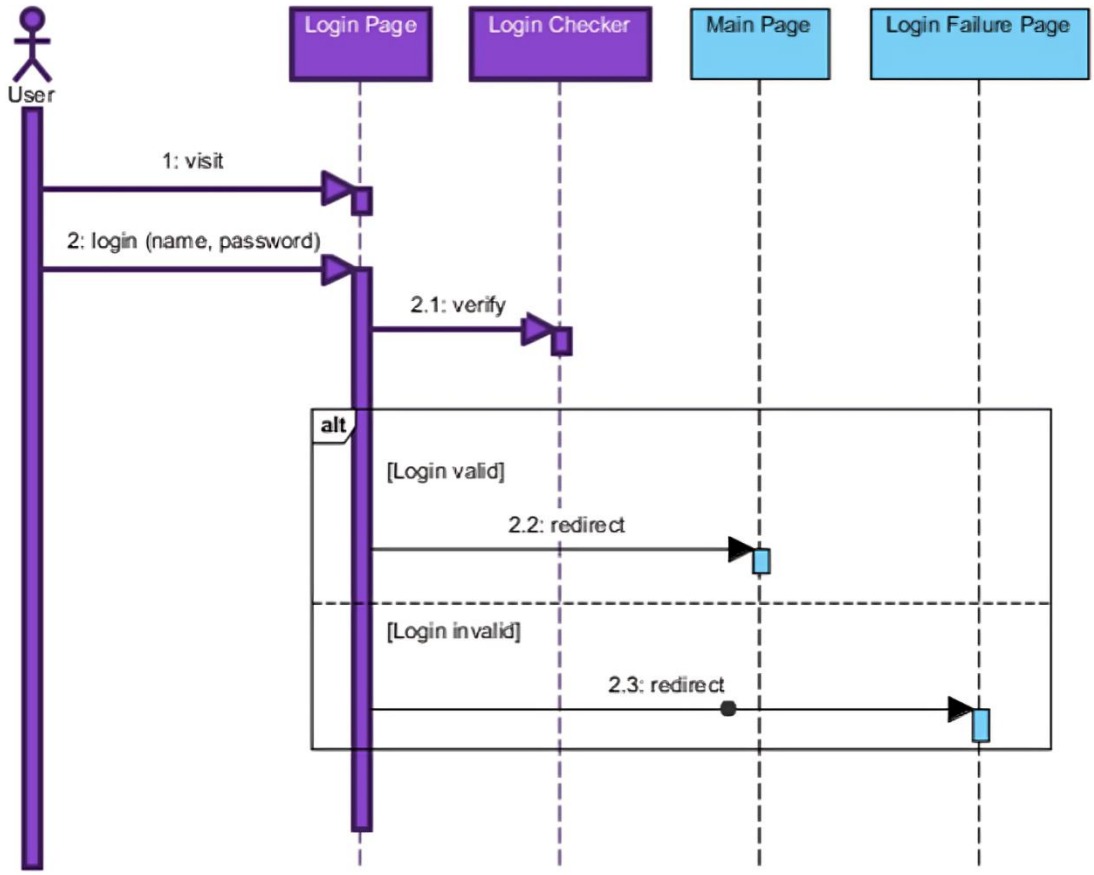

Figure 11 Sequence diagrams(Login)

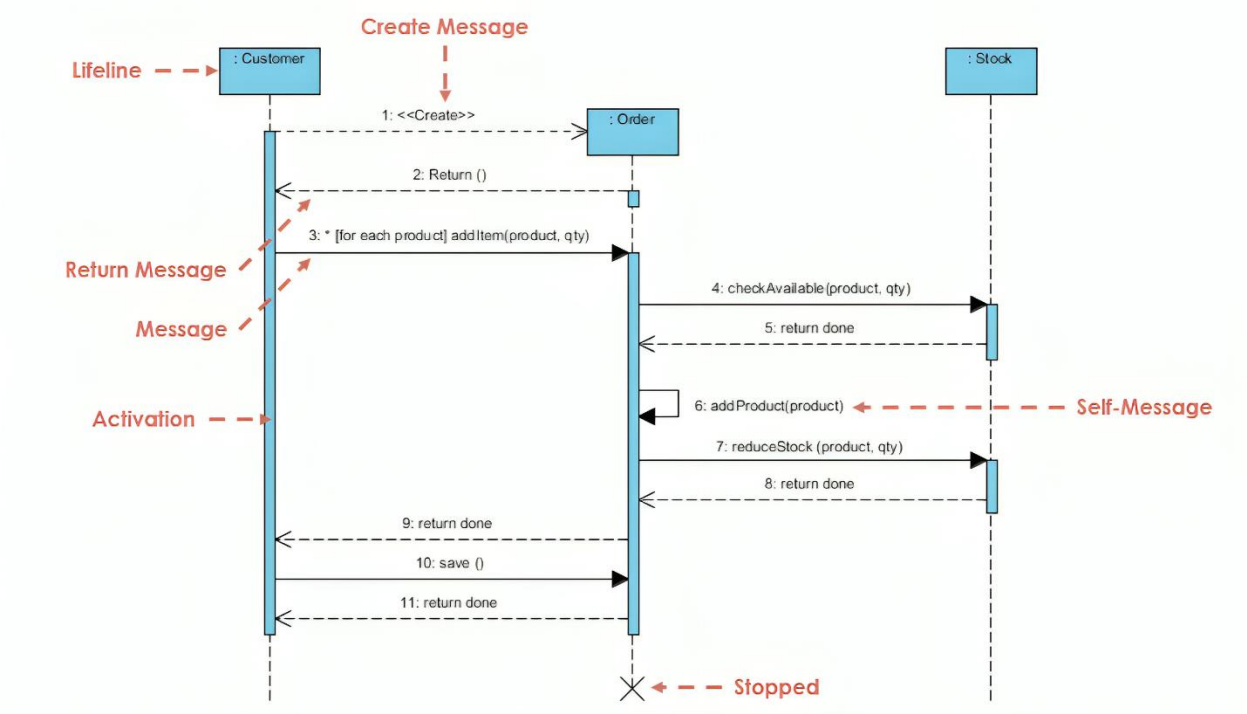

Figure 12 Sequence diagrams(Place order)

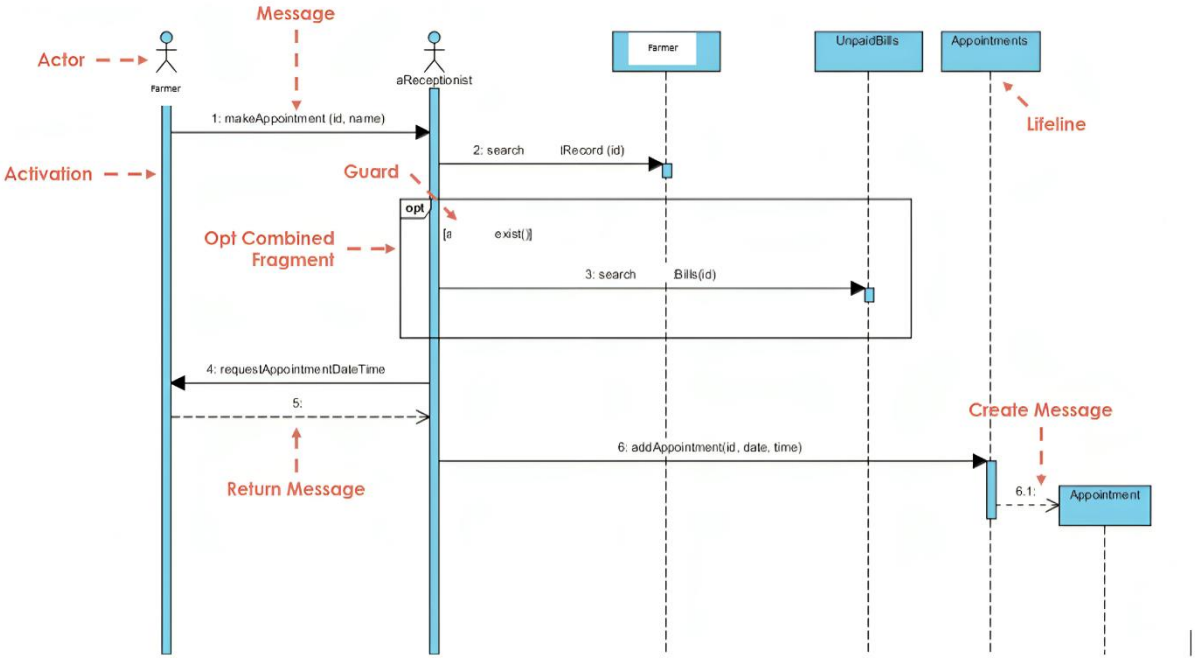

Figure 13 Sequence Diagram (consultation service)

4.5 Deployment View

Focus: The focus of this deployment view is on the interaction and communication between different modules and servers via APIs.

Viewpoint Type: architectural viewpoint.

Viewers: System architects, developers, and IT professionals.

Viewpoint Style: Package diagrams, Dependency diagrams, Module breakdowns, *Deployment diagram.*

Components: Modules/libraries for Computer with Web Browser and Client Module, Product Server, Bank Server.

Considerations: When designing this system, consider:

- **API Interactions**: Ensure seamless communication between components via APIs.
- **Security**: Protect sensitive data (e.g., payment information) during interactions.
- **Scalability**: Plan for system growth.
- **Performance**: Optimize API calls for efficiency.

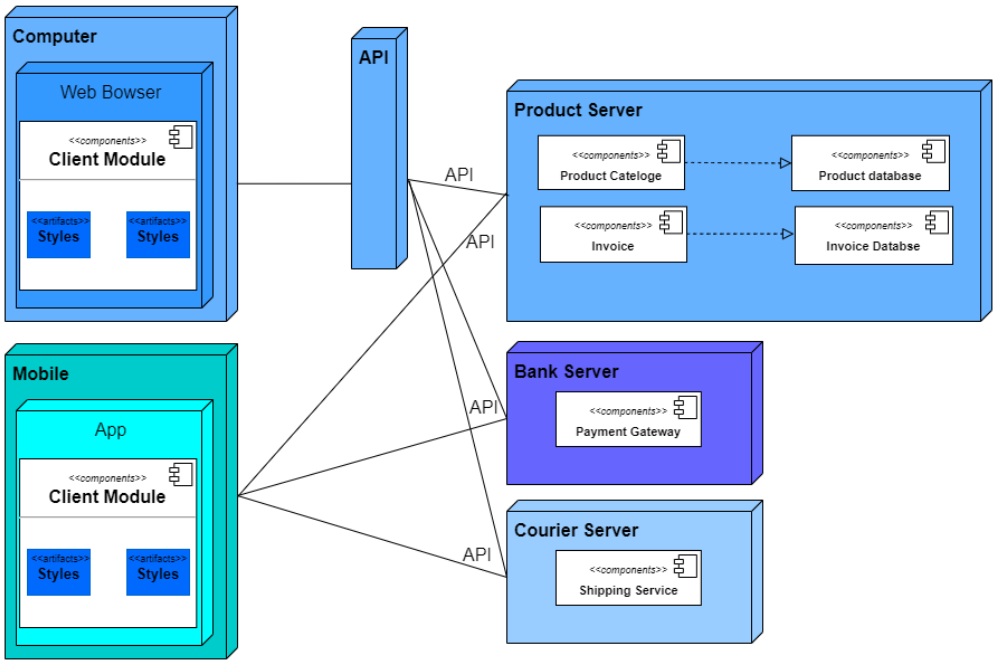

Figure 14 Deployment diagram

4.6 Implementation View

Focus: The focus of this implementation view is on the architecture and interaction of various components. It encompasses user interface components, client-side logic, middleware, authorization server, application server, database, and file storage system.

Viewpoint Type: This view falls under the **structural viewpoint** category.

Viewers: software engineers, system architects, and developers.

Viewpoint Style: **component** style viewpoint

Components: The diagram depicts several key components (e.g., User Interface Components, User Authentication Module, Client-Side Logic, Middleware, Authorization Server, Application Server, File Storage System, Database).

Considerations: Considerations for this implementation view might include:

- Ensuring security during user authentication and authorization processes.
- Efficient data storage and retrieval from the database.
- Effective client-server communication through middleware.

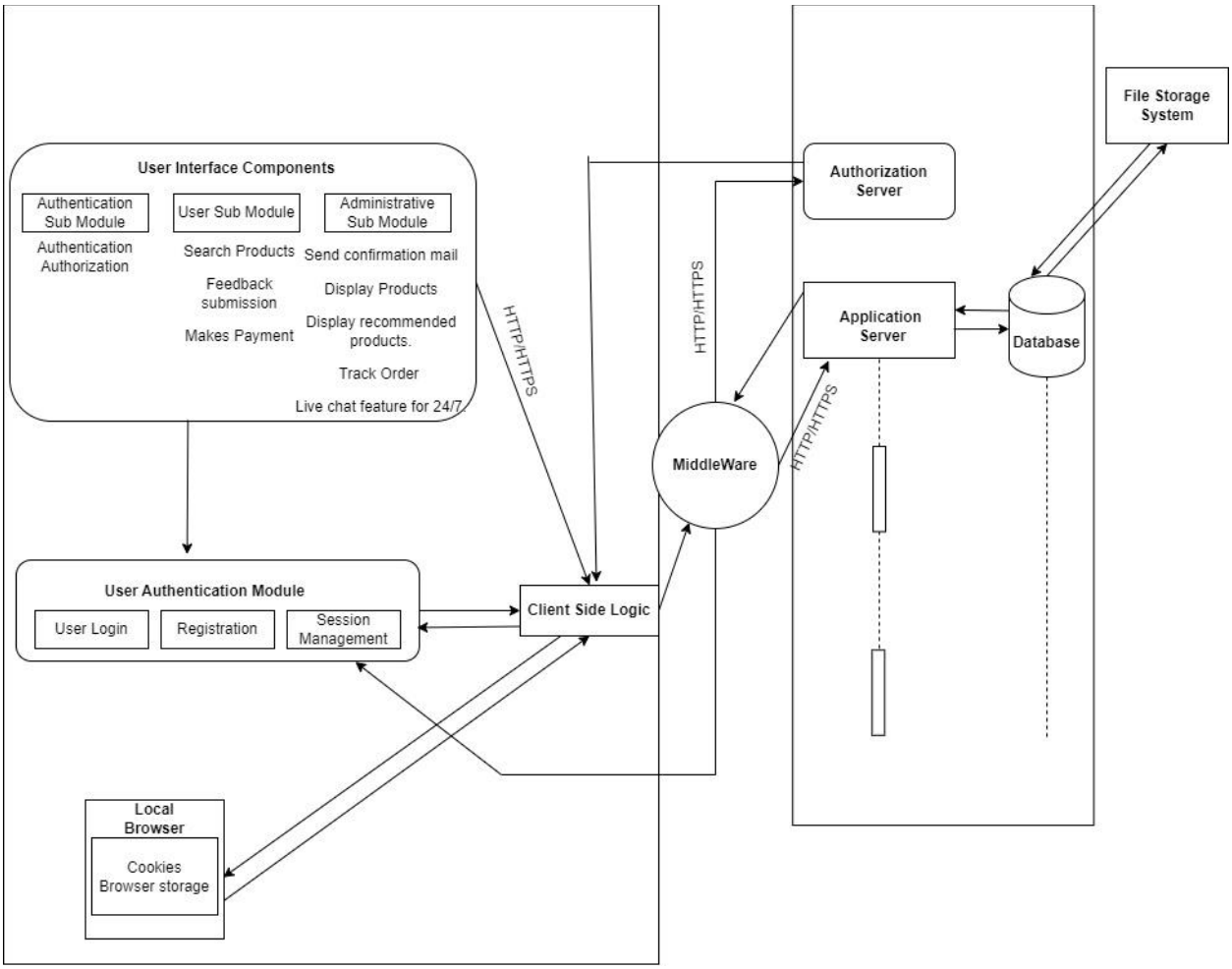

Figure 15 Implementation Diagram

5. Architectural Goals and Constraints

Scalability:

- E-Farm should be designed with scalability in mind to handle increasing user traffic and data volume, especially during peak seasons like harvesting or planting.
- Scalability can be achieved through horizontal scaling, where additional servers or resources are added to distribute the workload efficiently.
- Load balancing mechanisms should be implemented to distribute incoming requests across multiple servers evenly, preventing any single server from becoming overloaded.

Modularity:

- The architecture of E-Farm should be modular, meaning that it is composed of separate, interchangeable components or modules.
- Each module should have well-defined interfaces, allowing for easy integration of new features or updates without affecting the entire system.
- Modularity promotes code reuse, simplifies maintenance, and facilitates team collaboration by breaking down complex systems into manageable parts.

Flexibility:

- E-Farm should be flexible enough to adapt to changing market demands, user preferences, and technological advancements.
- The system should support multiple types of agricultural products, services, and user interactions, allowing for diverse offerings and user experiences.
- Flexibility also entails the ability to easily customize or extend the system to meet specific business requirements or regulatory changes.

Reliability:

- Reliability is essential for E-Farm to ensure consistent performance and availability, even in the face of failures or disruptions.
- The architecture should incorporate fault-tolerant design principles and redundant components to minimize the impact of hardware or software failures.
- Robust data backup and recovery mechanisms should be in place to protect against data loss and ensure data integrity.

Security:

- Security is a top priority for E-Farm to protect user data, financial transactions, and sensitive information from unauthorized access, data breaches, or cyber-attacks.

- The architecture should include robust authentication, authorization, and encryption mechanisms to safeguard user accounts and communications.
- Compliance with data privacy regulations such as GDPR or CCPA should be ensured to maintain user trust and legal compliance.

Performance:

- E-Farm should deliver optimal performance to provide users with fast response times and smooth user experiences.
- Performance optimization techniques such as caching, database indexing, and content delivery networks (CDNs) should be employed to minimize latency and improve throughput.
- Regular performance testing and monitoring should be conducted to proactively identify and address any bottlenecks or performance issues.

Maintainability:

- The architecture of E-Farm should be designed with maintainability in mind to facilitate ongoing maintenance, updates, and enhancements.
- Clear documentation, coding standards, and version control practices should be established to ensure code readability and consistency.
- Well-defined interfaces and separation of concerns should be maintained to isolate changes and minimize the risk of unintended side effects during maintenance activities.

Compatibility:

- E-Farm should be compatible with various devices, browsers, and platforms to accommodate diverse user preferences and technological environments.
- Compatibility testing should be performed across different devices, operating systems, and screen sizes to ensure consistent user experiences.
- The architecture should follow web standards and best practices to maximize interoperability and minimize compatibility issues.

Resource Limitations:

- E-Farm must operate within resource constraints such as limited server capacity, bandwidth, and memory.
- Resource utilization should be optimized through efficient algorithms, data compression techniques, and resource pooling to maximize available resources.
- Monitoring and capacity planning should be conducted regularly to anticipate resource needs and prevent resource exhaustion or performance degradation.

Regulatory Compliance:

- E-Farm must comply with relevant laws, regulations, and industry standards, particularly agriculture, data privacy, and e-commerce.
- Compliance measures should be implemented to protect user rights, ensure fair business practices, and mitigate legal risks.
- Regular audits and reviews should be conducted to verify compliance with applicable regulations and standards.

Cost:

- Cost-effectiveness should be considered throughout the design and implementation of E-Farm, balancing functionality, and performance with budgetary constraints.
- Cost-saving measures such as open-source technologies, cloud-based services, and scalable infrastructure should be leveraged where possible.
- Total cost of ownership (TCO) should be evaluated to assess the long-term affordability and sustainability of the system.

Legacy Systems Integration:

- If E-Farm needs to integrate with existing agricultural systems or databases, compatibility constraints may influence system design and implementation decisions.
- Integration points should be identified, and interoperability standards or protocols should be established to facilitate seamless communication between E-Farm and legacy systems.
- Data migration strategies and compatibility testing should be conducted to ensure smooth integration and minimize disruption to existing operations.

Appendices

A. Design Processes

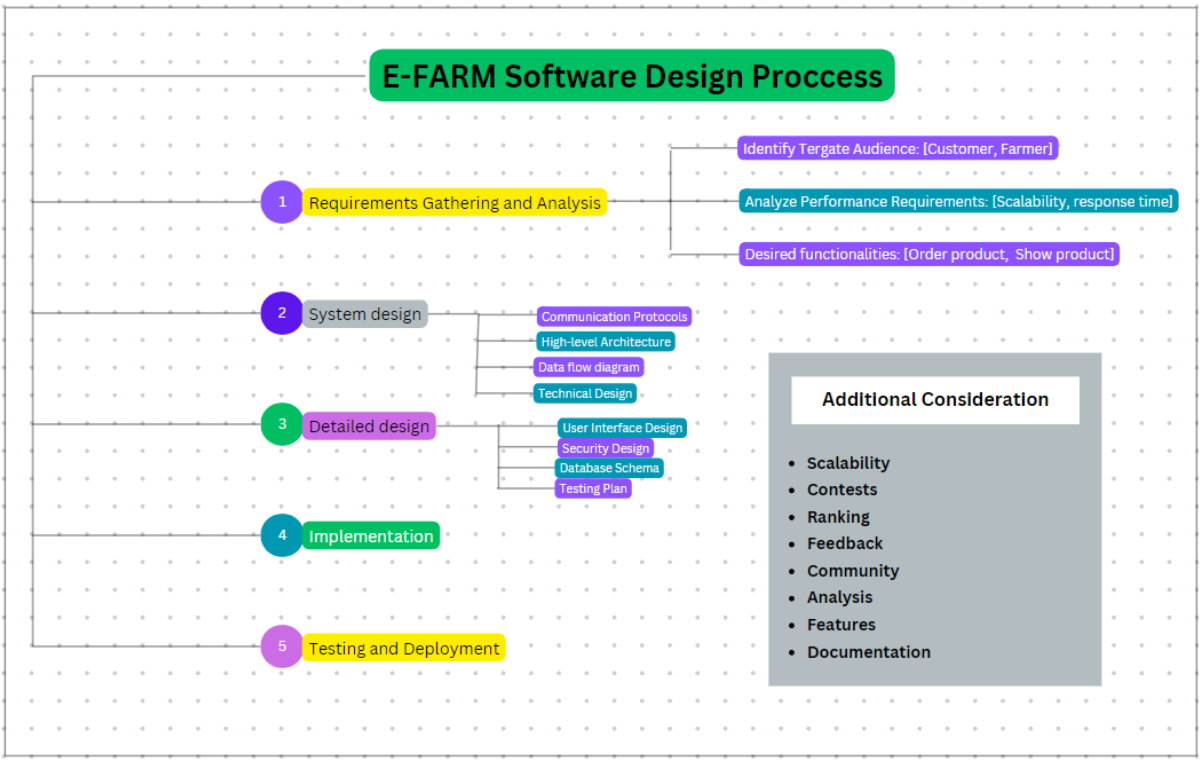

Figure 16 Design Processes

1. Requirements Gathering and Analysis:
    - Understand the target audience and their needs (Customer, Farmer).
    - Define functionalities desired, including order products, show product, etc)
    - Analyze performance requirements (scalability, response time).

2. System Design:

    - **High-level Architecture:** Define the main components (web application, product server, database) and their interactions.
    - **Technical Design:** Choose programming languages, frameworks, and databases based on requirements and expertise.
    - **Data Flow:** Describe how data moves between components, considering security and efficiency.
    - **Communication Protocols:** Define communication protocols between system components (API design).

3. Detailed Design:

    - **User Interface (UI) Design**: Create detailed mock-ups and prototypes for different user roles and functionalities.
    - **Database Schema**: Design a schema to store problems, submissions, user data, results, and other relevant information.
    - **Security Design**: Implement mechanisms to prevent unauthorized access, code injection, and other vulnerabilities.
    - **Testing Plan:** Define test cases for various functionalities and user interactions.

4. Implementation:
    - Develop the system based on the detailed design documents.
    - Use modular programming practices and version control systems.
    - Write unit and integration tests throughout the development process.

5. Testing and Deployment:
    - Conduct thorough testing for functionality, performance, and security.
    - Deploy the system to a stable environment with monitoring and logging capabilities.
    - Collect user feedback and iterate on design and implementation based on usage data.

Additional Considerations:

1. **Scalability**: Implement horizontal scaling strategies to handle large user bases and concurrent submissions.
2. **Contests and Ranking**: Design functionalities for creating, managing, and scoring contests with leaderboards and ranking systems.
3. **Feedback and Analysis:** Integrate features for providing feedback to users on their submissions and offering code analysis tools.
4. **Community Features**: Consider forum or chat functionalities for users to interact and learn from each other.
5. **Documentation:** Create comprehensive documentation for users, developers, and administrators.

B. Architecture Patterns

Client-Server

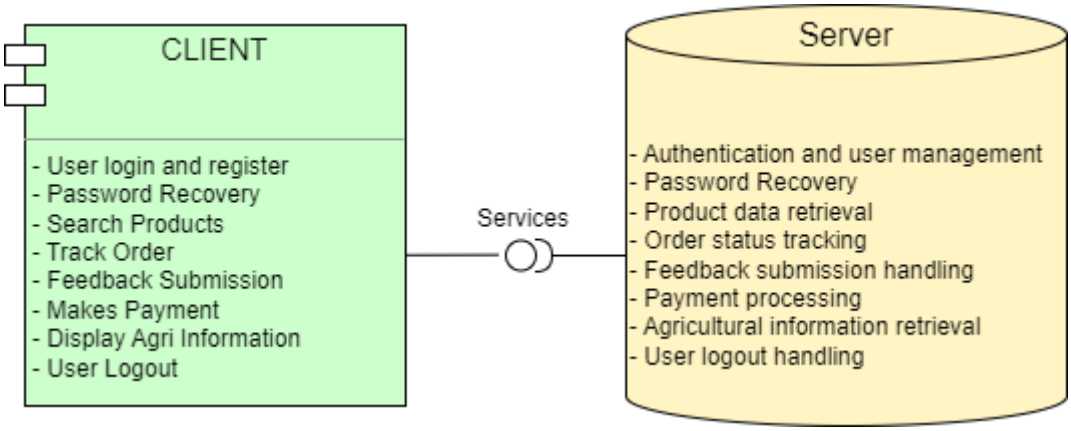

Figure 17 Client-Server Pattern

The client-server pattern in software design and architecture is a model where a client component initiates requests for services or resources, and a server component processes these requests, executes business logic, and manages data, fostering modular and scalable system design.

- **User login and register**: The client and the server (authentication and user management).
- **Password Recovery**: The client (user) and the server (authentication) for password recovery.
- **Search Products**: The client interacts with the server to search and retrieve product data.
- **Track Order**: The client interacts with the server to track order status.
- **Feedback Submission**: Involves communication between the client and server.
- **Makes Payment** communicates with client and server for payment processing.
- **Display Agri Information**: The client interacts with the server to display agricultural information.
- **User Logout**: Involves user interface (client) interacting with the server to log out.

Model-View-Controller (MVC)

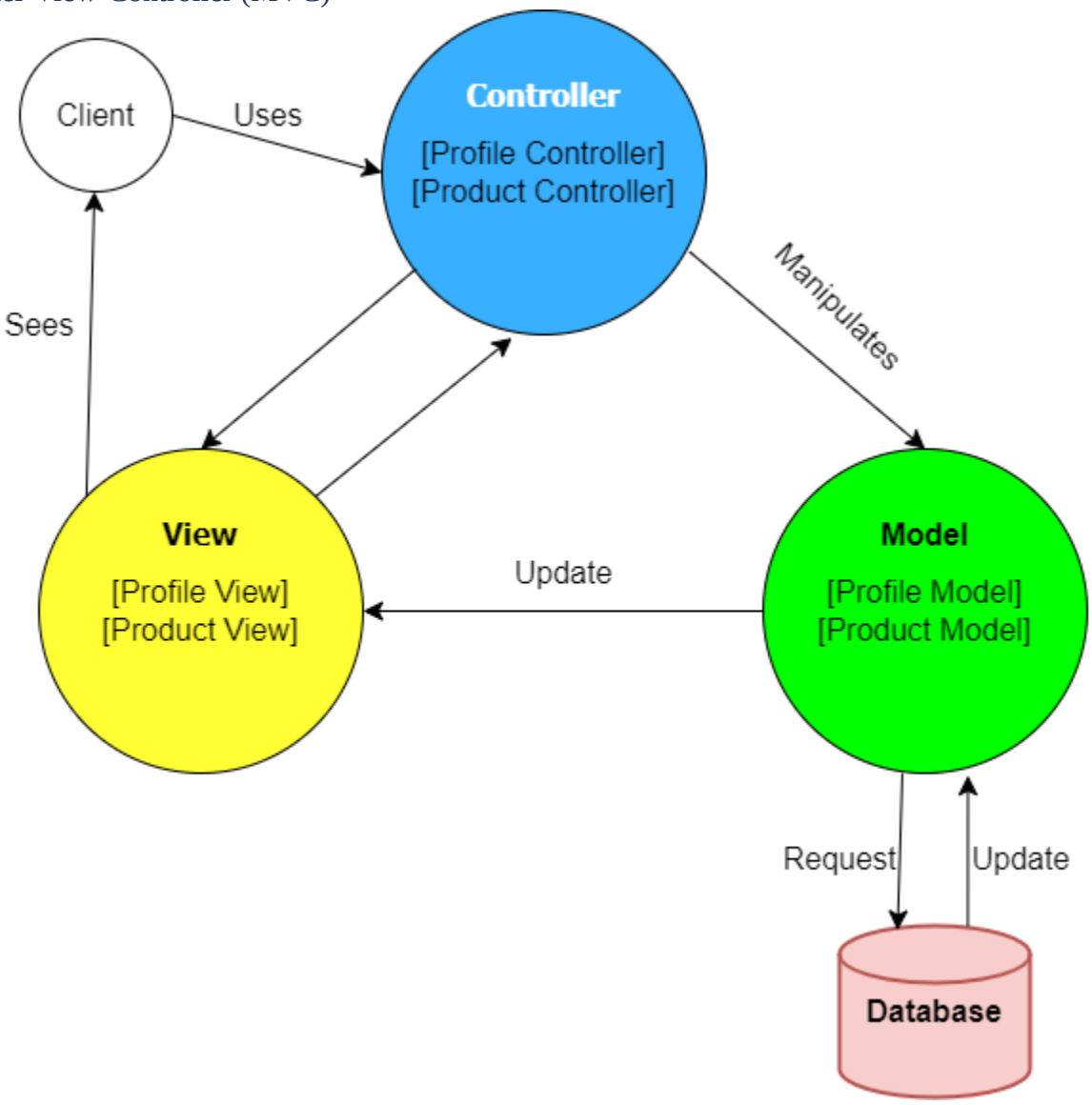

Figure 18 Model-View-Controller (MVC) Pattern

The client-server pattern in software design and architecture is a model where a client component initiates requests for services or resources, and a server component processes these requests, executes business logic, and manages data, fostering modular and scalable system design.

- **Update Profile**: MVC separates the user interface, business logic, and data storage, making it suitable for handling profile updates.
- **Display Products Availability**: MVC can manage the separation of data, business logic, and user interface.

Layered Pattern

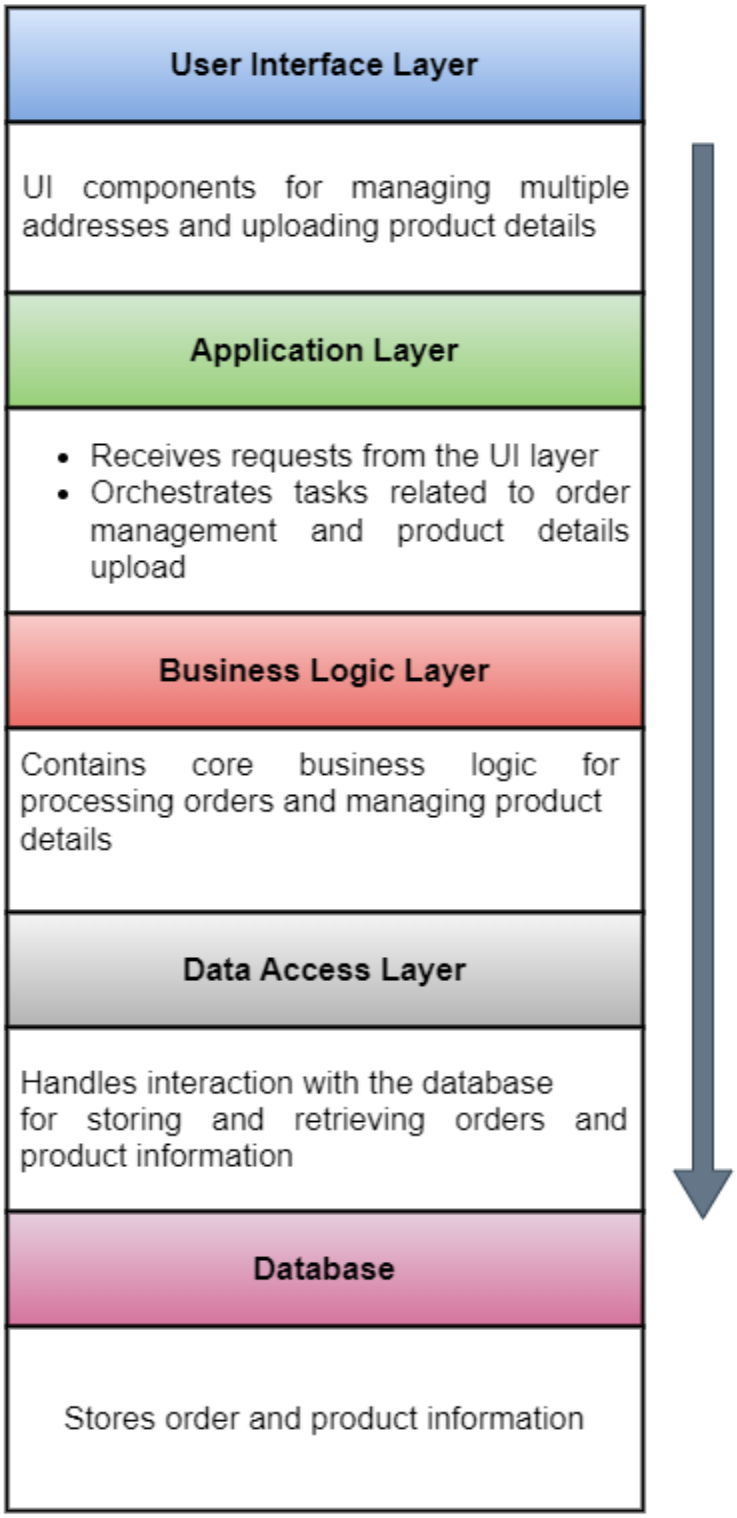

Figure 19 Layered Pattern

The layered pattern in software design and architecture involves organizing components into distinct layers, each responsible for a specific set of functionalities, promoting separation of concerns and modularity.

- **Order Multiple Address**: Layered pattern allows for a clear separation of concerns, suitable for managing multiple addresses in the order process.
- **Upload product details**: Layered pattern allows for clear separation of concerns, suitable for managing the upload and storage of product details.

Blackboard Pattern

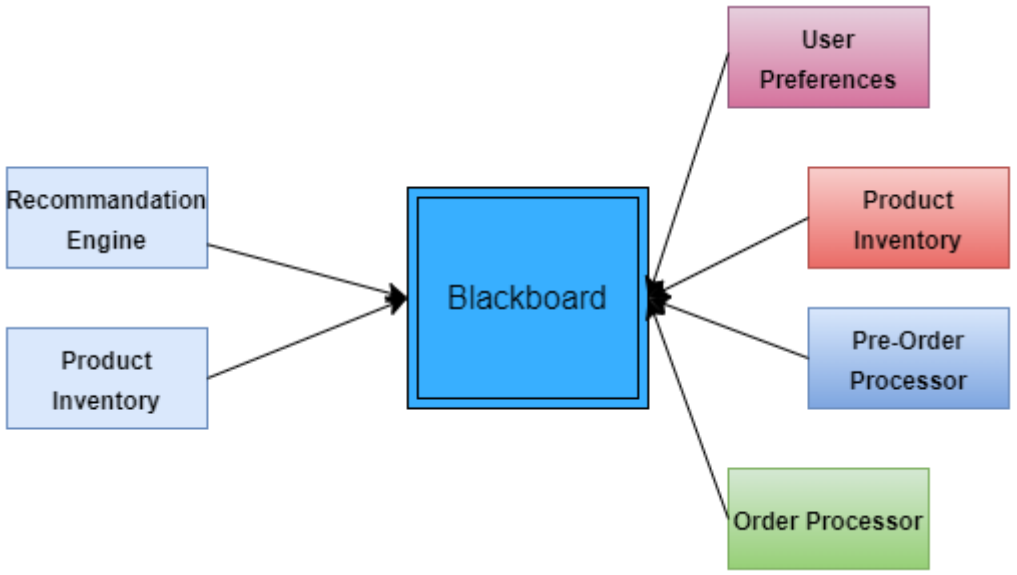

Figure 20 Blackboard Pattern

The Blackboard pattern in software design and architecture is a model where multiple independent components contribute their knowledge to a shared repository (the blackboard), which is then used to solve complex problems or make decisions.

- **Display Product Recommendation**: The Blackboard pattern can be used for collaborative decision-making, which is suitable for generating and displaying recommendations.
- **Pre-Order Request**: The Blackboard pattern can be used for collaborative decision-making, which is suitable for handling pre-order requests.

Pipe-Filter

Figure 21 Pipe-Filter Pattern

The pipe-filter pattern in software design and architecture is a model where a series of components (filters) process data in a linear sequence through interconnected pipes, facilitating modularity and reusability in complex systems.

- **Receive confirmation mail**: This pattern can be used to process and send confirmation emails as a series of filters.
- **Filter Products**: Filters can be applied on the server to process and filter product data based on user preferences.
- **Providing Ratings and Reviews**: Filters can process and manage ratings and reviews before displaying them.