Architectural pattern & View

for

E~Farm

Prepared by

MD ASIF MAHMUD | MUH2025004M

RUBYA RASHED | MUH2025014M

IMTIAZ CHOWDHURY | MUH2025027M

MAHAMUDUL HASAN TALUKDER | MUH2025028M

MEHEDI HASAN | MUH2025032M

Institute of Information Technology

Noakhali Science and Technology University

Group Assignment for

SE 3211 ~ Software Design and Architecture

Submitted to

Dipok Chandra Das

Assistant Professor

Institute of Information Technology

Noakhali Science and Technology University

Date of Submission: 07 December 2023

Architectural Description

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

An architectural description in software engineering refers to a comprehensive representation of a system's architecture. It serves as a detailed document that outlines the structure, components, interactions, and behaviour of a software system from an architectural perspective. This description is crucial for understanding, communicating, and maintaining the system's design throughout its lifecycle.

View vs Viewpoint

Viewpoints:
Viewpoints are perspectives or vantage points from which the architecture of a system is examined or analyzed. Each viewpoint represents a specific concern or set of concerns, allowing stakeholders to comprehend different aspects of the system. Viewpoints serve as lenses through which the architecture is viewed, enabling focused discussions and analysis.

Views:
Views are the representations or visualizations derived from viewpoints. They provide a structured and organized depiction of the architecture based on specific concerns. Each view is tailored to address a particular set of concerns and stakeholders' needs. Views make the architecture more accessible and understandable by presenting relevant information in a clear and concise manner.

Viewpoint, view & architectural description

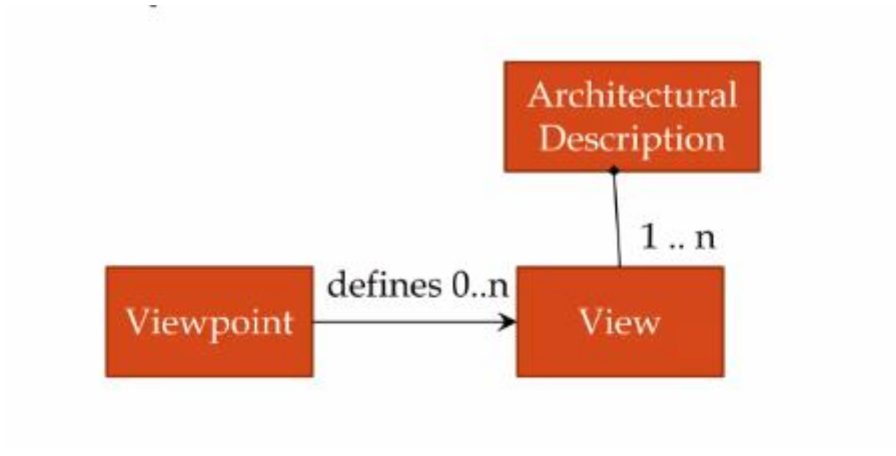

Fig:1(source- Internet)

•An architectural description is a collection of one or more views.

•A viewpoint is used to define (the structure and content of) zero or more views.

•The structure and content of a particular view is defined by exactly one viewpoint.

View type and Style

View types:

View types refer to different perspectives or concerns of a system that stakeholders might have. Each view type represents a distinct set of concerns and provides a specific viewpoint for understanding the architecture.

Common View types Include:

Functional View: Focuses on the system's functionality, depicting how distinct functions or features interact.
Logical View: Emphasizes the logical structure of the system, including modules, components, and their relationships without concern for physical deployment.
Development View: Addresses aspects related to software development, such as code organization, modules, libraries, and dependencies.
Process View: Concentrates on the dynamic behaviour of the system, illustrating interactions, tasks, and processes.
Physical View: Describes the physical deployment of the system, including hardware, networks, servers, and their connections.

Styles:

Architectural Styles represent common patterns or design paradigms used to organize and structure systems. These styles dictate the fundamental principles and constraints that shape the architecture.

Examples of Architectural Styles:

Client-Server: Separates the client interface from the server that manages resources, allowing for scalability and centralization of resources.
Microservices: Comprises small, independent services that communicate through APIs, promoting flexibility and scalability.
Layered Architecture: Organizes components into horizontal layers, each responsible for specific functionalities, promoting modularity and maintainability.
Event-Driven Architecture: Systems communicate through events, enabling loosely coupled and asynchronous interactions.
Service-Oriented Architecture (SOA): Encapsulates functionalities into services that can be accessed and reused across the system.

Importance:

Clarity and Communication: View types and styles provide a structured way to communicate various aspects and design decisions of the system architecture to stakeholders.
Consistency and Standards: They establish standard approaches for organizing and representing system architectures, ensuring consistency across documentation.
Problem-Specific Solutions: Different view types and styles allow architects to address specific concerns or problems more effectively during system design.

Microservices Architecture:

Description: Focuses on breaking down a system into a collection of small, independently deployable services, each serving a specific business goal.
Characteristics: Emphasizes decentralization, independence, and scalability of services.

Approach: Views the system as a set of individual microservices that communicate via APIs or lightweight protocols.
Purpose: Aims for flexibility, rapid development, and scalability of services.

Service-Oriented Architecture (SOA):

Description: Organizes software components as loosely coupled services that can be reused for different purposes.
Characteristics: Focuses on service reusability, standard interfaces, and interoperability between disparate systems.
Approach: Views the system as a set of services with well-defined interfaces that communicate via standardized protocols.
Purpose: Aims for flexibility, reusability, and interoperability across different components and systems.
While they are not view models themselves, these architectural styles can be represented using different architectural viewpoints or views within a comprehensive architectural description. For instance, within the 4+1 architectural view model or ISO/IEC/IEEE 42010 standard, you can describe the individual components, interactions, deployment, and other aspects specific to a Microservices Architecture or a Service-Oriented Architecture. These views help in comprehensively understanding and communicating the intricacies of these architectural styles within a larger system.

Microservices Architecture:

Why: E-Farm involves various services like the marketplace, equipment rental, and consultancy, each with its unique functionalities.
Benefits: Scalability, independent deployment, and maintenance of services. It allows flexibility in development and enables teams to focus on specific services.
Considerations: Requires well-defined service boundaries, necessitates robust communication between services, and might add complexity in managing distributed systems.

Service-Oriented Architecture (SOA):

Why: Like microservices, SOA focuses on service components but with a broader approach to interoperability and reuse.

Benefits: Encourages reusability of services, promotes loose coupling between components, and facilitates integration with external systems.
Considerations: Requires a clear understanding of service contracts and standardized interfaces.

Layered Architecture:

Why: Well-suited for systems with different tiers of functionalities, like presentation, business logic, and data storage (common in many applications).
Benefits: Separation of concerns, easy maintenance, and scalability. Each layer handles specific tasks, promoting modularity.
Considerations: Might become complex if not properly organized, and changes in one layer might affect others.

Event-Driven Architecture:

Why: Useful when handling multiple interactions and events within the system, such as notifications, alerts, or real-time data processing.
Benefits: Allows asynchronous communication, scalability, and flexibility in handling various events.
Considerations: Proper event handling and management are crucial; it might require robust event processing mechanisms.

Recommendation:

Given the diverse nature of services in E-Farm and the need for scalability and flexibility, a combination of Microservices Architecture or Service-Oriented Architecture (SOA) might be particularly suitable. This approach would allow you to segregate different services, maintain them independently, and potentially scale each service according to demand. However, the final choice should be based on a detailed analysis of your specific requirements, technical capabilities, and the team's familiarity with the selected architectural model.

4+1 model within our project E-farm

The 4+1 architectural view model is a framework that provides multiple views to describe and understand software architecture. In the context of your project, E-Farm, here's how the 4+1 model can be applied:

Logical View:
Purpose: Describes the functionality of the system without detailing its implementation.
Representation: Use class diagrams, package diagrams, or component diagrams.
In E-Farm: Display the logical structure of the platform, highlighting components like user interfaces, databases, and core functionalities such as marketplace, equipment services, and agro solutions.

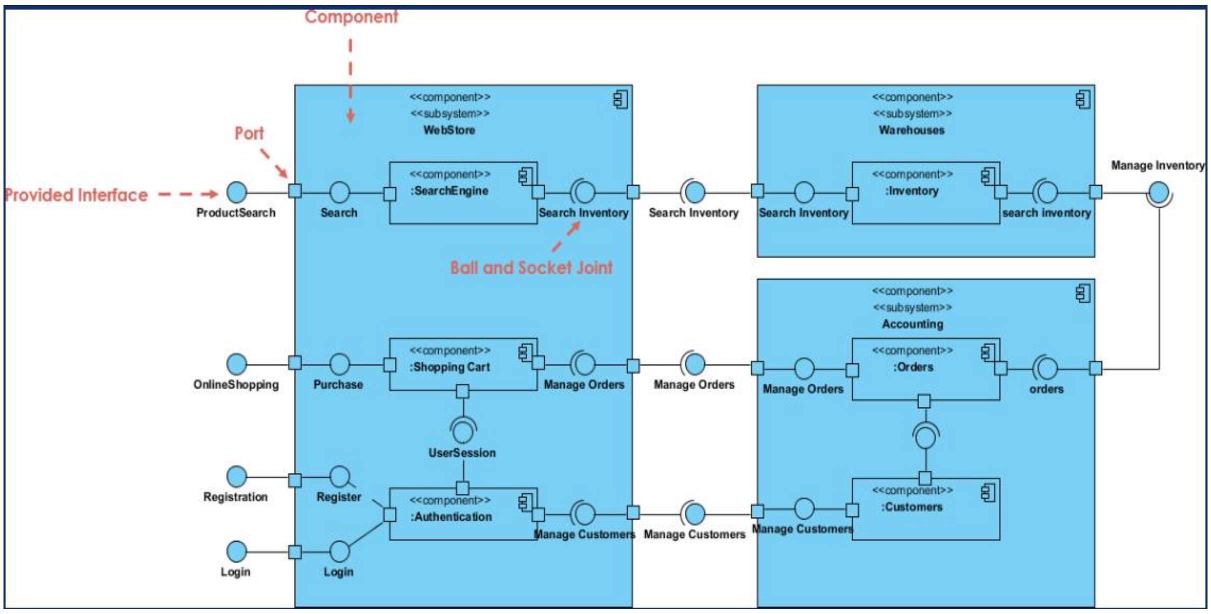

Fig:2(source- Internet)

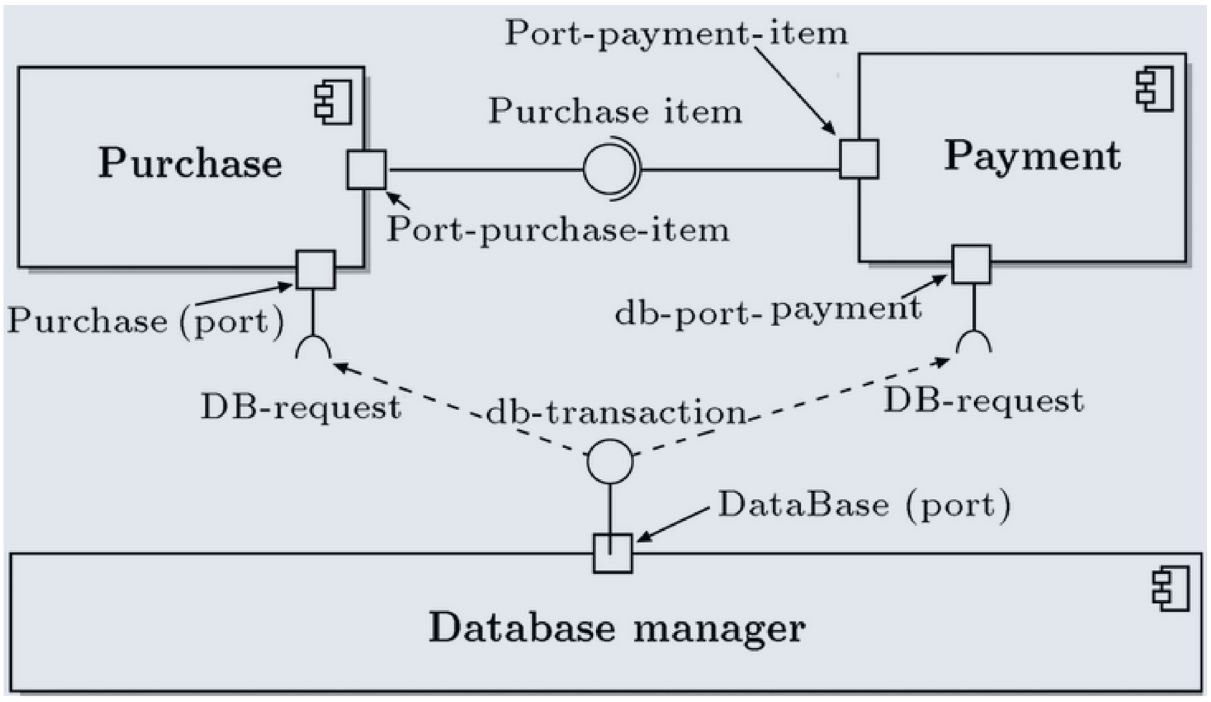

Fig:3(source- Internet)

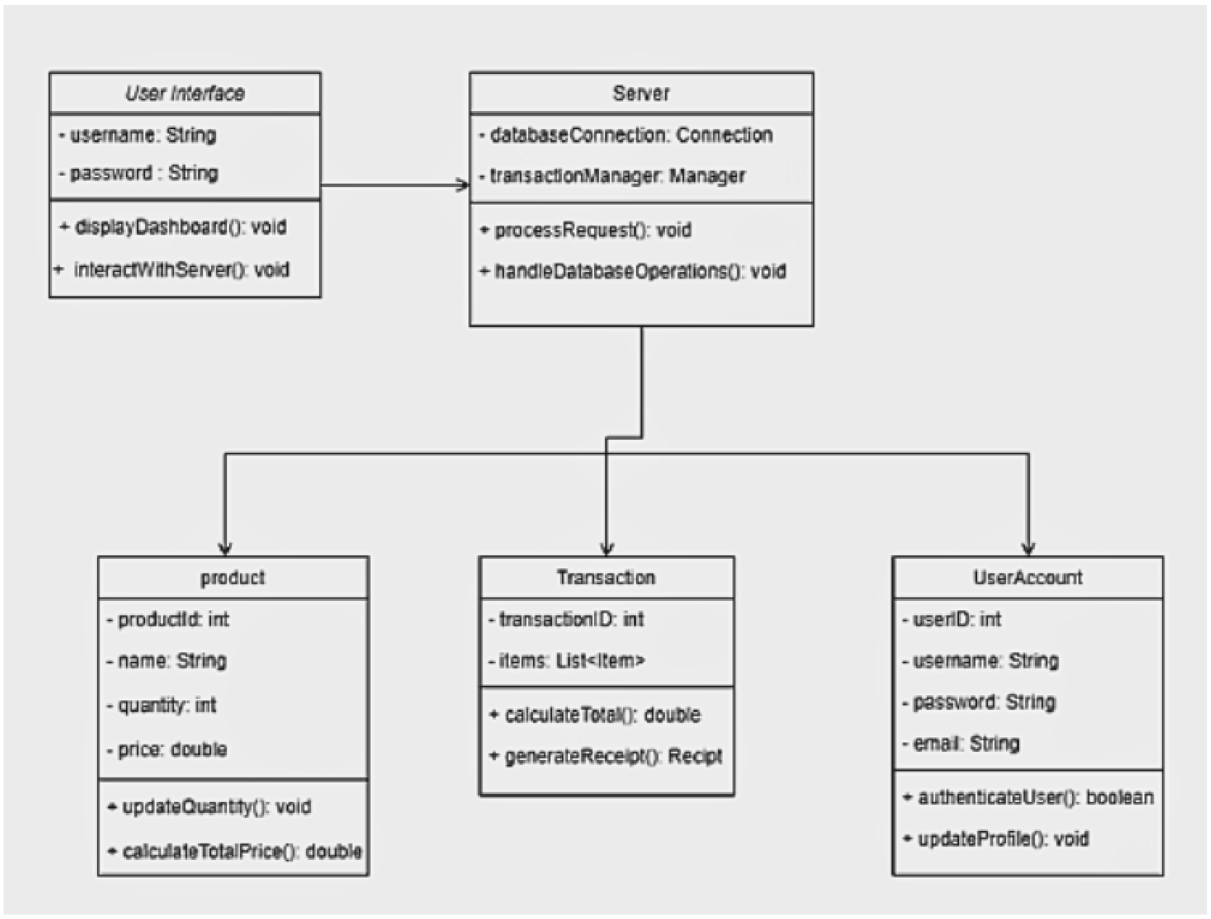

Fig:4

Development View:
Purpose: Focuses on the software's development aspect, modules, and organization.
Representation: Use package diagrams, dependency diagrams, or module breakdowns.
In E-Farm: Present how the software is structured for development, showcasing modules or libraries used for different functionalities like user authentication, data processing, and interface components.

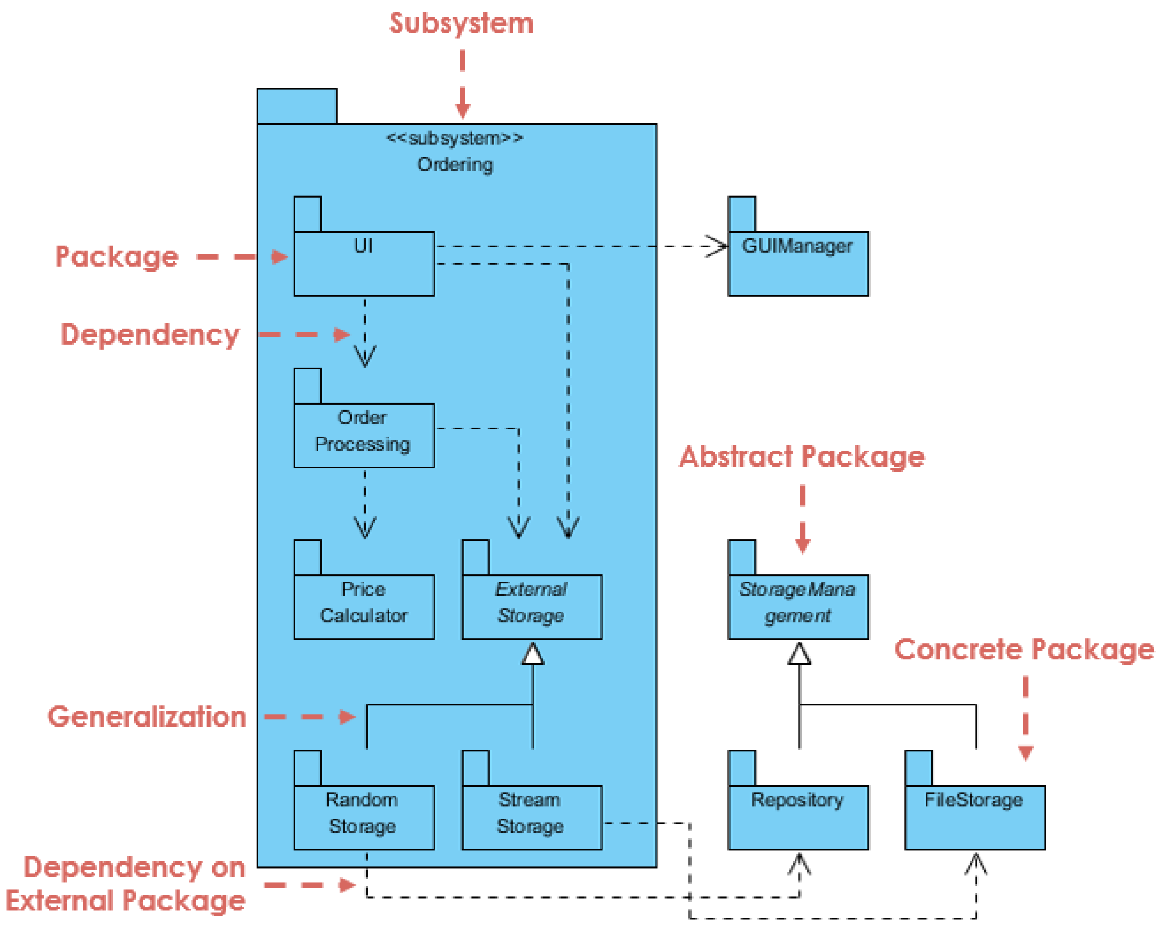

Fig:5(source- Internet)

Process View:

Purpose: Illustrates the system's dynamic behavior and interactions among processes. ***Representation***: Use sequence diagrams, activity diagrams, or state diagrams.
In E-Farm: Display the flow of interactions between different system components, such as how a user places an order, how equipment rental is initiated, or how consultation services are provided.

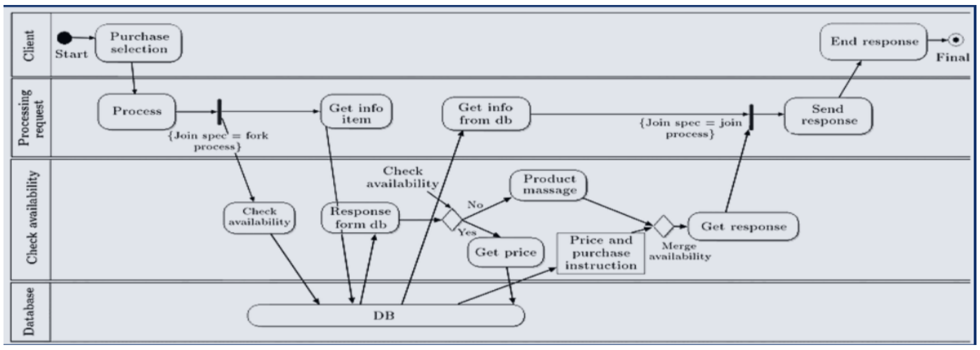

Fig:6

Physical View:
Purpose: Shows the physical architecture and deployment aspects.
Representation: Use deployment diagrams, infrastructure diagrams.
In E-Farm: Outline how the system is deployed physically, indicating servers, databases, network configurations, and how different components are distributed across the infrastructure.

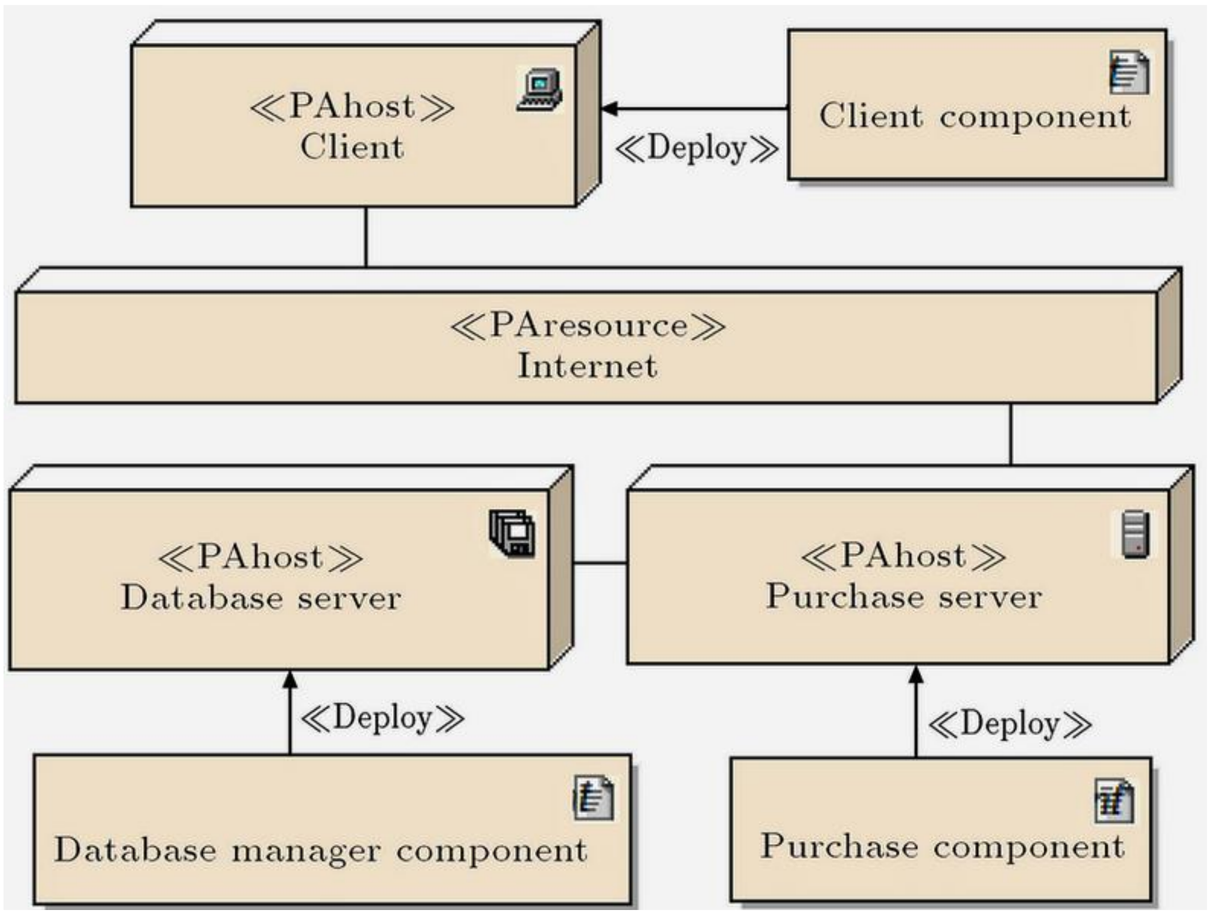

Fig:7(source- Internet)

+1 - Use Case or Scenarios:
Purpose: Illustrates specific scenarios or use cases to understand system functionality. ***Representation***: Use case diagrams, sequence diagrams.
In E-Farm: Showcase scenarios like a farmer listing a product, a consumer purchasing, equipment rental, or a consultation request to demonstrate how users interact with the system. Importance in E-Farm: Communication: Using the 4+1 model helps in communicating various aspects of E-Farm's architecture to different stakeholders. Understanding: It provides multiple perspectives for understanding the system, catering to the needs of developers, architects, managers, and users. Documentation: This model aids in comprehensive documentation of the system's architecture, promoting easier maintenance and future enhancements. Applying the 4+1 model to E-Farm allows for a holistic representation, ensuring a thorough understanding of its architecture from different viewpoints and perspectives.

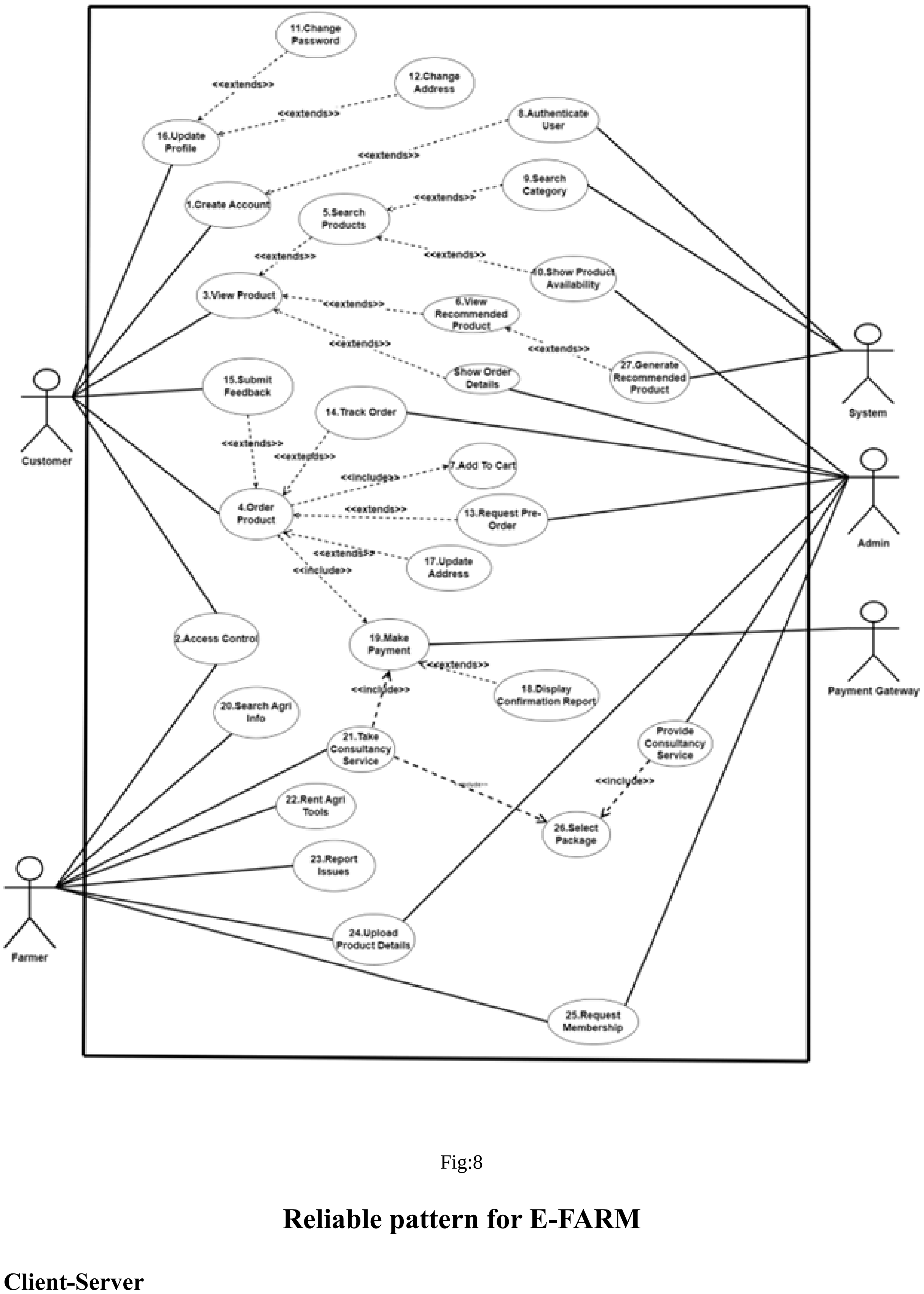

Fig:8

Reliable pattern for E-FARM

Client-Server

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**: communicate 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)

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.

Pipe-Filter

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.

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.

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.