Internal Assessment Assignment

Name:-Ranjeet Choudhary
PRN:-202301100046
Branch:-Software Engineering

1. Identification and Comparison of Architectural Style

The GreenBasket system is a Farmer-to-Consumer marketplace that facilitates product listing by farmers, product browsing and purchasing by customers, and administrative monitoring by platform administrators. Major platform features include:

- Authentication and Role Management
- Product Management for Farmers
- Product Browsing, Cart, and Searching for Buyers
- Orders, Payments, and Subscriptions
- Reviews and Messaging
- Administrator Monitoring and Reporting

The system must support **scalability, security, availability, high performance, and fault tolerance**.

After evaluating multiple architectural styles, **Microservices Architecture** is identified as the most suitable style for GreenBasket.

---

2. Architectural Pattern Identified and Justification

Identified Architectural Pattern

Microservices-based Architecture Pattern

Justification (with case-specific examples)

GreenBasket contains different feature domains, each having different users and traffic patterns. Microservices allow each domain to exist as an independent service that can scale and evolve without affecting the rest of the system.

a) Scalability Based on Functional Loads

Different platform modules receive different traffic volumes.

Functional Requirement Group	**Expected Traffic**	**Microservice**
Product Browsing & Cart (FR-6 to FR-8 / REQ-19 to REQ-24)	Very High	Catalog/Cart Service
Orders & Payments (REQ-25 to REQ-30)	Moderate → Peak during offers	Order/Payment Service
Authentication (REQ-1 to REQ-9)	High	Authentication Service

With microservices, **only high-load services scale**, reducing cost.
A monolithic system would require scaling the entire application.

b) Maintainability and Continuous Deployment

Services can be updated and deployed independently.

- Example: A change in **stock alert logic (REQ-16, REQ-17)** affects only the Product Service.
- Example: A new **Product Management feature (FR-3)** can be deployed without affecting Orders or Payments.

In a monolithic model, such changes require redeploying the entire system, causing downtime.

c) High Performance with Parallel and Asynchronous Processing

During checkout:

- Validate Cart
- Process Payment (REQ-25–27)
- Update Inventory (REQ-18)
- Generate Invoice (FR-10)

With microservices, these steps run **asynchronously across multiple services**, reducing response time.

d) Security and Role-Based Isolation

Sensitive modules are isolated.

Security Requirement	Responsible Microservice
Password Encryption & Role Access (REQ-3 to REQ-7)	Authentication Service
Secure Payment (REQ-25 to REQ-27)	Payment Service
Reviews & Messaging (REQ-30 to REQ-33)	Review Service

Even if one service is compromised, critical modules remain secure.

e) Resilience and Fault Isolation

Failure in one module does not stop the entire platform.

Example: If **Reviews & Messaging Service** goes down, users can still:

- Log in
- Browse products
- Add to cart
- Place orders and make payments

In monolithic architecture, failure of one module could crash the entire application.

f) Technology Flexibility

Each microservice can use a different stack:

Requirement	Recommended Technology
Real-time messaging	WebSockets
Product search/filter	Search indexing engine
Payments	PCI-compliant secure infrastructure

This flexibility is not possible with monolithic or layered architecture.

Mapping of Platform Features to Microservices

System Feature	Corresponding Microservice
Authentication & Role Access	Authentication Service
Product Management	Product Service
Product Browsing & Cart	Catalog/Cart Service
Orders, Payments & Subscriptions	Order/Payment Service
Reviews & Messaging	Review & Messaging Service
Administrative Reports & Monitoring	Admin Service

Comparison of Architectural Styles

Criterion	Microservices Architecture	Layered Monolithic Architecture
Scalability	Individual Service Scaling	Full System Scaling
Deployment	Independent deployments	Complete redeployment
Fault Isolation	High	Low
Technology Choice	Flexible	Limited
Development Speed (Initial)	Moderate	Fast
Long-term Maintainability	High	Low
Database Management	Distributed, Independent DBs	Centralized, Single DB

Why Layered Monolithic Architecture Was Not Selected

Although easy for initial development, it is not suitable for the long-term needs of GreenBasket due to:

- Tight coupling between modules as functionality grows
- Need to redeploy entire system for any change
- No option for independent scaling for high-load modules like search and checkout
- Failure in one module can bring down the entire platform

Conclusion

Microservices Architecture is the most appropriate architectural style for GreenBasket.
It supports the platform's need for high scalability, performance stability, security isolation, fault tolerance, continuous deployment, and technology flexibility.

While a layered monolithic architecture enables fast initial development, it becomes inefficient and risky as GreenBasket grows. Therefore, the **Microservices Architecture Pattern is the most suitable and recommended style for this project.**