SECTION 5 — Monolithic vs Microservices Architecture

And How Our Monsta Backend Is Microservice-Ready

1. What Is Software Architecture?

Software architecture defines:

- How different parts of an application are structured
- How components communicate
- How responsibilities are separated
- How the system scales

Choosing the right architecture affects:

- Maintainability
- Scalability
- Deployment strategy
- Team collaboration
- Long-term system stability

Two common backend architectures are:

1. Monolithic Architecture
2. Microservices Architecture

2. What Is a Monolithic Architecture?

A monolithic architecture means:

The entire backend application runs as a single unified system.

All features exist in one codebase and run as one service.

Example structure:

```
Single Server
 ├── Auth
 ├── Products
 ├── Orders
 ├── Payments
 ├── Admin
```

All modules are deployed together.

Advantages of Monolithic Architecture

- Simple to develop initially
- Easy to deploy
- Easier debugging (single system)
- Less infrastructure complexity
- Suitable for small to medium projects

Disadvantages of Monolithic Architecture

- Harder to scale individual features
- Entire application must be redeployed for small changes
- Large codebases become difficult to maintain
- Tight coupling between modules

3. What Is Microservices Architecture?

Microservices architecture means:

The application is divided into independent services.

Each service:

- Has its own responsibility
- Can have its own database
- Can be deployed independently
- Communicates with other services through APIs

Example:

- Auth Service
- Product Service
- Order Service
- Payment Service
- Notification Service

Each runs independently.

Advantages of Microservices

- Independent scaling
- Independent deployment
- Better fault isolation
- Suitable for large teams
- Technology flexibility per service

Disadvantages of Microservices

- Infrastructure complexity
- Network communication overhead
- Deployment complexity
- Requires DevOps maturity
- Harder debugging across services

Deployment	Single unit	Independent services
Scaling	Entire app	Individual services
Complexity	Lower	Higher
Infrastructure	Simple	Complex
Best For	Small/medium apps	Large-scale systems

5. What Is a Modular Monolith?

There is a third practical approach:

Modular Monolith

This means:

- Application runs as a single deployment
- But internally structured into clean modules
- Each module has clear boundaries
- Business logic is isolated

It combines:

- Simplicity of monolith
- Structure of microservices

This is the architecture we are using.

6. How Monsta Is Architected

Our Monsta backend is:

A Modular Monolith that is Microservice-Ready

It runs as:

- One Node.js application
- One database connection
- One deployment

But internally structured as:

```
modules/
 ├── identity/
 ├── governance/
 ├── supply/ (future)
 ├── demand/ (future)
```

Each module:

- Has its own controllers
- Has its own services
- Has its own models
- Has minimal cross-dependency

7. Why Monsta Is Microservice-Ready

Even though it runs as a monolith, it is structured in a way that:

- Each module can be extracted into a separate service later.
- Business logic is not tightly coupled.
- Communication is clean.
- Shared utilities are centralized.

Example:

The identity module could become: **Auth Service**

The governance module could become: **Governance Service**

Minimal refactoring would be required.

8. Why We Chose Modular Monolith for Students

For learning purposes:

- Microservices infrastructure is complex.
- Students should first master clean architecture.
- Domain separation is more important than distributed deployment.

By building:

- Clear modules
- Clean controllers
- Separate services
- Centralized middleware

Students learn scalable thinking.

9. Architectural Principles Used in Monsta

Monsta follows:

1. Separation of concerns
2. Layered architecture
3. Domain-based modularization
4. Middleware-driven security
5. Config-driven setup
6. Utility-based reusable logic

This ensures:

- Code readability
- Scalability
- Maintainability
- Future service extraction

10. How a Microservice Extraction Would Work

Suppose in future we want to extract Identity:

Steps:

1. Copy identity module into new repository.
2. Add independent server.js.
3. Connect to its own database.
4. Expose only identity APIs.
5. Communicate via HTTP or events.

Because the module is isolated, extraction is possible.

That is what “microservice-ready” means.

11. Why Architecture Matters for Students

Students often focus only on writing endpoints.

Enterprise development focuses on:

- Structure before features.
- Scalability before shortcuts.
- Isolation before complexity.

Architecture defines how far the system can grow.

12. Summary

By the end of this section, students should understand:

- What monolithic architecture is
- What microservices architecture is
- Why microservices are powerful but complex
- What a modular monolith is
- Why Monsta follows modular monolith
- How Monsta is structured to be microservice-ready

This sets the mindset for writing scalable backend systems.