Layered Architecture Pattern: An Overview

The Layered Architecture Pattern organizes complex applications into distinct layers, each with a specific role. This presentation explores the pattern's benefits, common layers, communication styles, and best practices, providing a comprehensive understanding of this popular architectural approach.

by Mohamed Amr

What is the Layered Architecture Pattern?

The Layered Architecture Pattern structures an application into horizontal layers. Each layer performs a specific role and resides within a hierarchy. Key principle is separation of concerns. Layers are loosely coupled, enhancing modularity. **Strict Layering:** Each layer can only call the layer directly below. **Relaxed Layering:** Layers can call any layer below. Example: Presentation, Business, Data Access.

Key Benefits of Layered Architecture

- Modularity & separation of concerns are enhanced.
- Maintainability and testability are improved.
- Code reusability is increased across the application.
- Clear organization and understandability.
- Individual layers can be deployed and scaled independently.

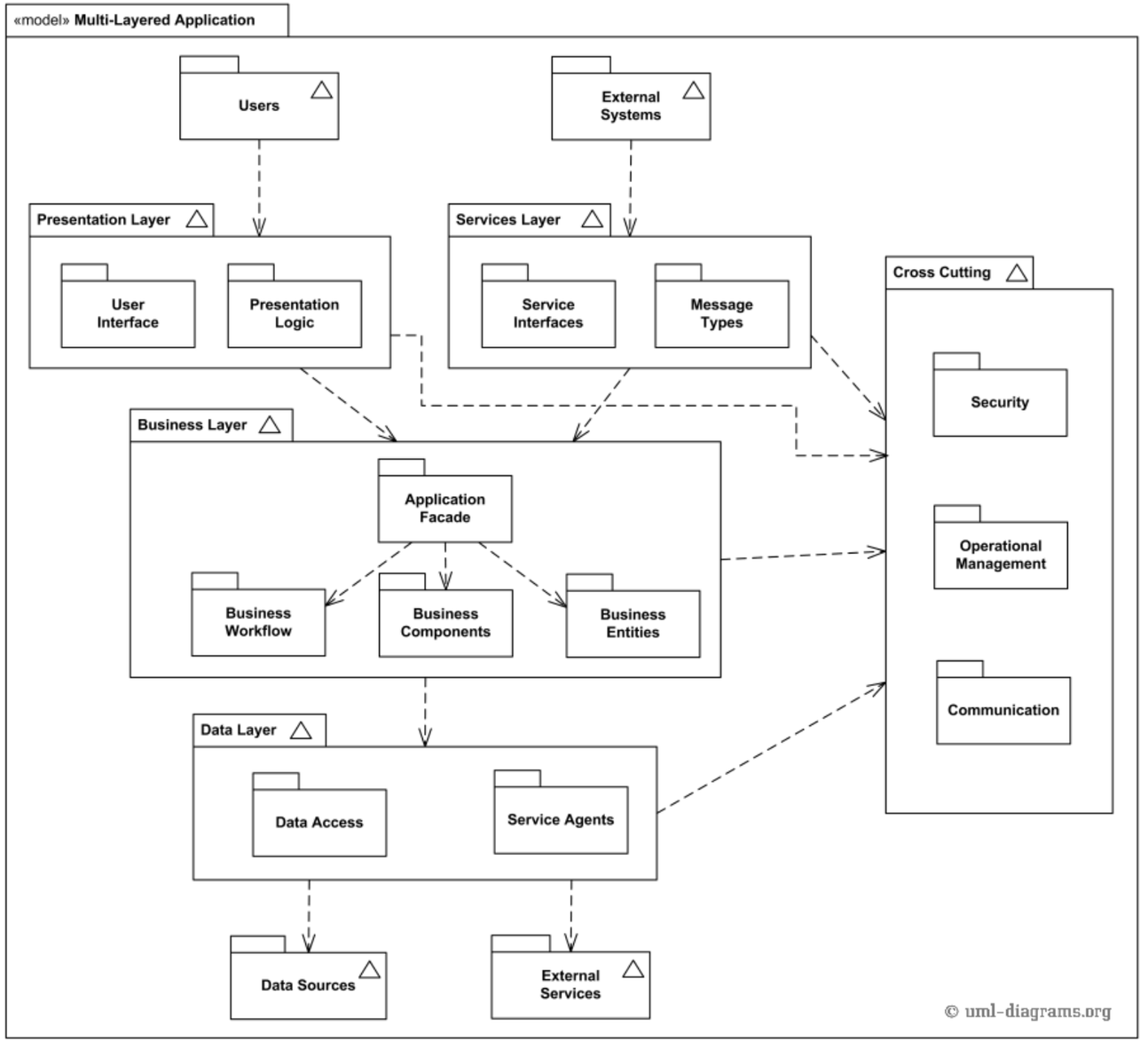
Common Layers in Layered Architecture

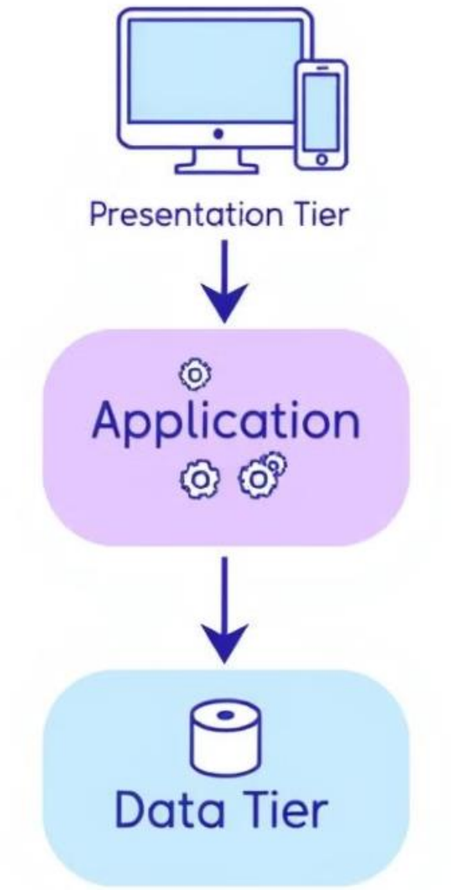
Presentation Layer (UI): Handles user interactions and presents data.

Business Logic Layer: Implements business rules and workflows.

Data Access Layer: Interacts with databases or other data sources.

Persistence Layer (Optional): Maps business objects to database records.


«model» Multi-Layered Application
Users
External Systems
Presentation Layer
User Interface
Presentation Logic
Services Layer
Service Interfaces
Message Types
Cross Cutting
Security
Business Layer
Application Facade
Business Workflow
Business Components
Business Entities
Operational Management
Communication
Data Layer
Data Access
Service Agents
Data Sources
External Services
© uml-diagrams.org



Types of Layered Architecture: Three-Tier

Presentation Tier Application Tier Data Tier Simple to implement, the three-tier architecture comprises a presentation tier, an application tier, and a data tier. It's best suited for small to medium-sized applications. However, it can become monolithic over time.

Three-Tier: Pros, Cons, Scenarios

- Pros:
  - Fast development (single codebase).
  - Easy to deploy (one application).
  - Clear separation of concerns (UI, logic, data).
- Cons:
  - Single point of failure (app server downtime).
  - Difficult to scale individual components.
  - Changes require redeployment of entire app.
- Scenario: simple e-commerce site (MVP, limited functionality).
- Recommended: when fast time-to-market is priority.

Three-Tier Architecture Example

Presentation Tier

React component displaying user data.

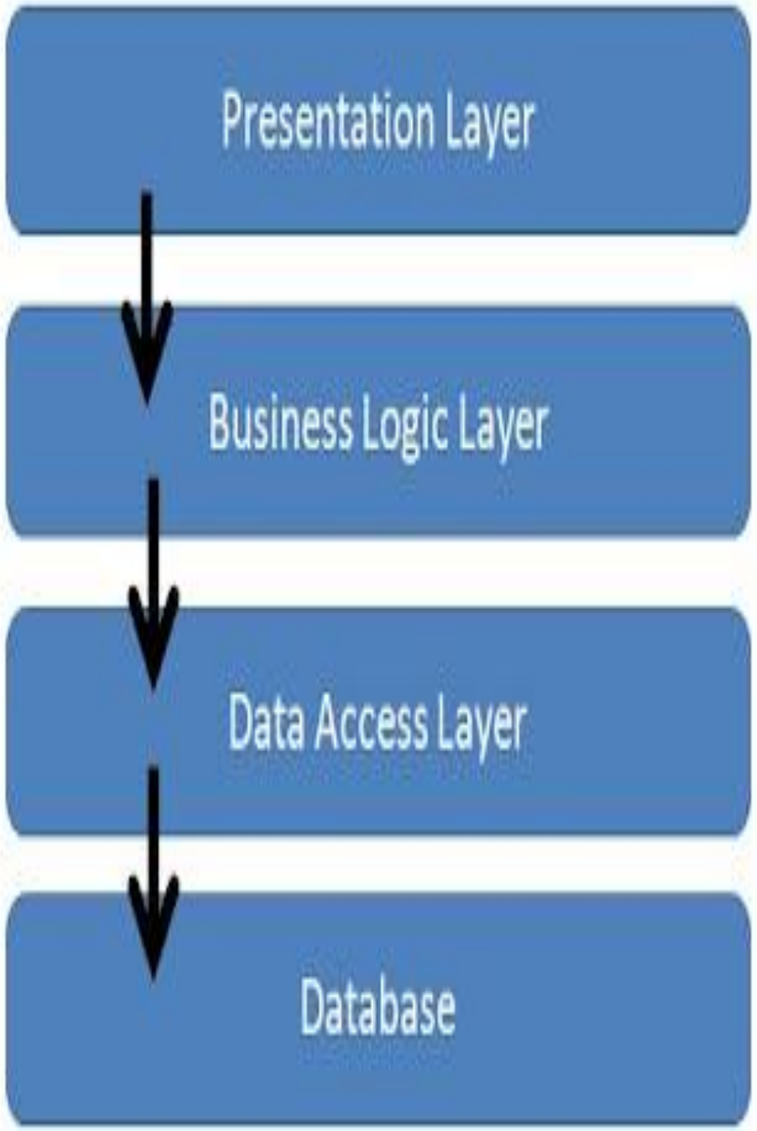
Application Tier

Node.js/Express API endpoint.

Data Tier

MongoDB storing user data.

Communication is synchronous using a request-response model.



Types of Layered Architecture: Four-Tier Architecture

Presentation Tier Application Tier Business Tier Data Tier Four-Tier Architecture offers a greater separation of concerns. It includes a Presentation, Application, Business, and Data tier. This enhances scalability and maintainability but introduces more complexity.

Four-Tier: Pros, Cons, Scenarios

- Pros:
  - Improved scalability (separate application/logic layers).
  - Enhanced security (dedicated business logic layer).
  - Easier maintenance (modular design).
- Cons:
  - Increased complexity (multiple layers and codebases).
  - Higher development cost (specialized expertise required).
  - Risk of over-engineering (unnecessary complexity for simple apps).
- Scenario: complex banking application (high scalability, security).
- Recommended: when application is large and requires high level of security.

Four-Tier Architecture Example

Presentation

React component for product data.

Application

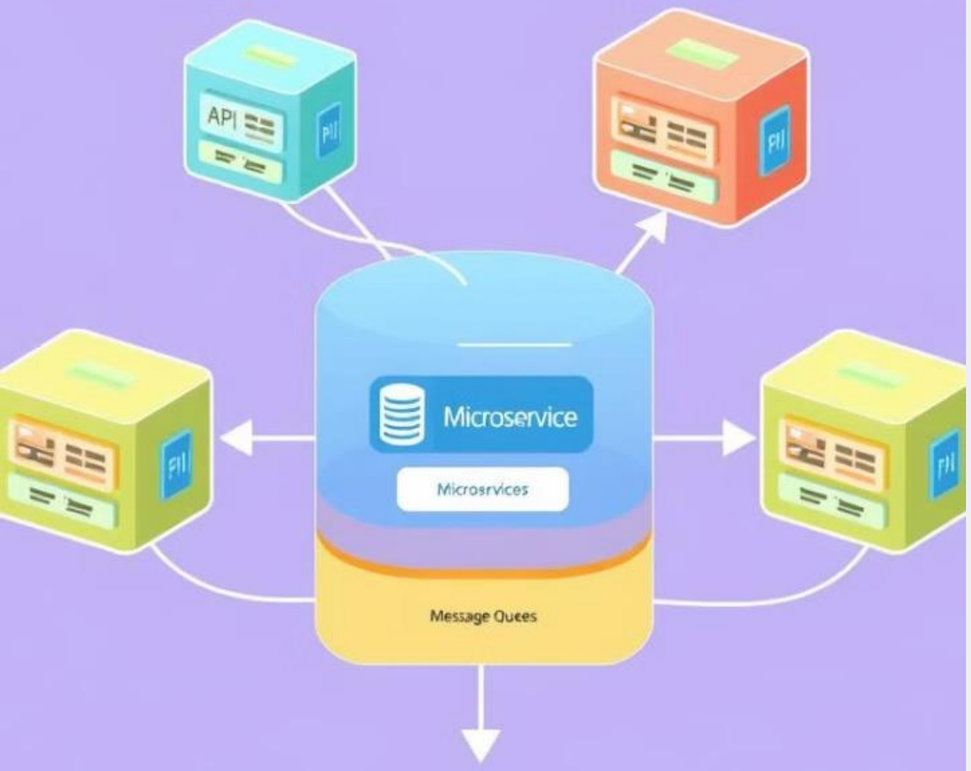
Node.js/Express API endpoint.

Infrastructure

Database access and services.

Domain

Product business rules.



Types of Layered Architecture: Microservices (Hybrid)

Each microservice follows layered architecture internally. This allows for individual deployment and scalability. Examples include Authentication Service, Product Catalog Service. Significant architectural complexity is introduced.

Microservices with Layered Architecture: Pros, Cons, Scenarios

- Pros:
  - Highly scalable (independent deployment of services).
  - Increased resilience (failure isolation).
  - Technology diversity (different stacks for each service).
- Cons:
  - Increased complexity (distributed system management).
  - DevOps expertise required (containerization, orchestration).
  - Potential data inconsistency (distributed databases).
- Scenario: Netflix-style streaming platform (huge scale, continuous updates).
- Recommended: when you have a team of experts to handle scaling.

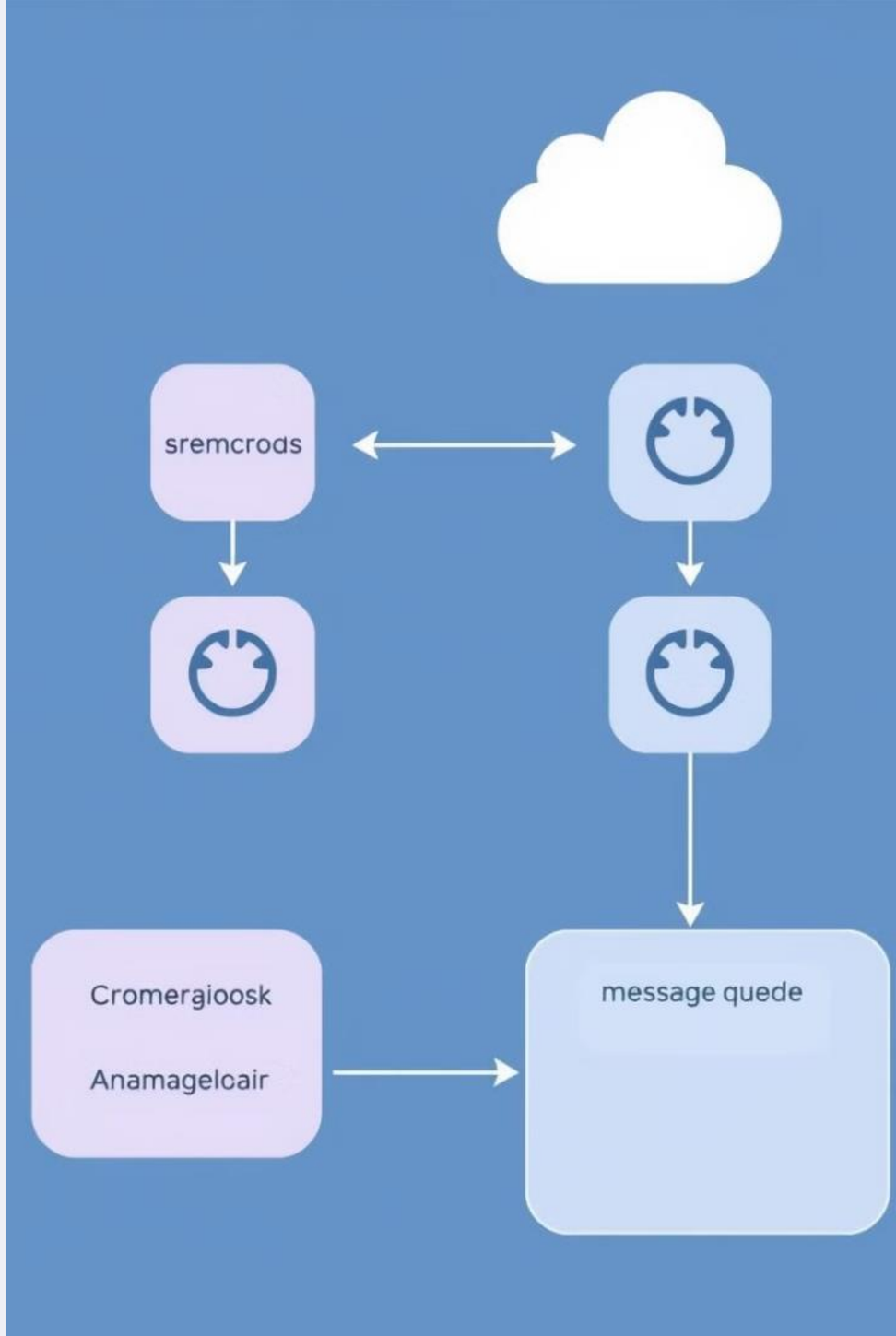
When to Use Layered Architecture

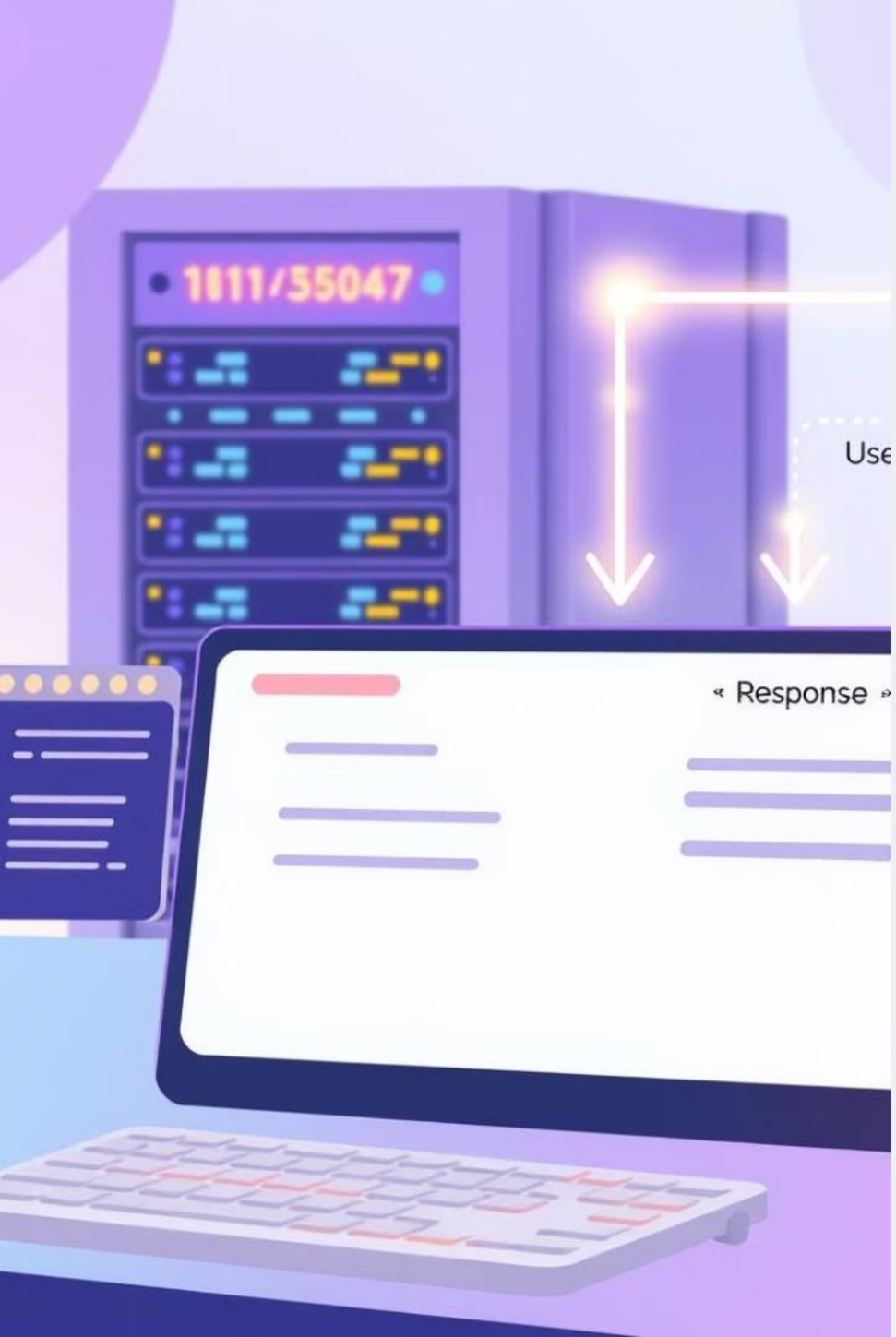
- Three-Tier: simple web applications, quick development needed
- Four-Tier: complex enterprise applications, scalability and security are critical
- Microservices: large-scale systems, independent scaling and deployment are essential

Communication Between Layers

Synchronous: Uses direct calls (blocking). Examples: HTTP requests.

Asynchronous: Employs message queues (non-blocking). Examples: Apache Kafka. Trade-offs involve performance, reliability, and complexity. Choose the appropriate communication method based on the specific layer interactions.





Synchronous Communication:

The Request-Response Model : Communication where the sender waits for a response from the receiver. - Mechanisms: REST APIs, GraphQL, direct function calls. - Example: A UI layer calling an API to fetch data.

Asynchronous Communication: The Event-Driven Model - Definition: Communication where the sender doesn't wait for a response. - Mechanisms: Message queues (Kafka, RabbitMQ), WebSockets. - Example: Microservices communicating via message queues.

The Sinkhole Anti-Pattern

What is it?

A layer simply passes requests to the next layer without adding value.

The Problem

It reduces performance and increases complexity with no benefit.

Example

A controller directly calls a database query without logic.

Avoiding the Sinkhole

Add Value

Apply business logic in each layer.

Avoid Unnecessary Layers

Refactor to remove redundant layers.

Optimize Data Access

Cache to reduce database load.

Monitor

Implement logging to detect behavior.

Best Practices for Layered Architecture

- Define clear layer responsibilities.
- Minimize dependencies between layers.
- Implement well-defined interfaces.
- Enforce layering constraints.

When to Use Layered Architecture?

The Layered Architecture Pattern is suitable for applications with well-defined domains. It excels in projects requiring high maintainability and scalability and is crucial where separation of concerns is essential. Not ideal for very small or exceptionally complex systems.

Key References

3

Core Books

"Patterns of Enterprise Application Architecture" by Martin Fowler, "Clean Architecture" by Robert C. Martin.

2

Cloud Docs

Microsoft Azure and AWS documentation on application and microservice design.

+100

Online Articles

Blog posts on layered architecture best practices.

References :

- https://www.geeksforgeeks.org/layered-architecture-in-computer-networks/

- https://medium.com/@sagar.hudge/layers-in-software-architecture-c8cc16329ff6

-https://www.sciencedirect.com/topics/computer-science/layered-architecture

Code:

-https://github.com/MohamedAmr23/Layered-Architecture