Your Guide to System Architecture

Ngane Emmanuel

2024-09-27

Contents

1 Introduction

Building software systems isn't all about diving right into the coding aspect of it. There are numeroussteps that need to be completed before a single line of code can be written. In Agile methodology, the designing phase is a very important phase as this gives us a detailed understanding of how the system will look like.

In software design, there are high and low level designs. In this handbook, we will focus on the high level design which involve system architecture.

We are going to look at various system architectures used in software engineering. Each architecture is explained in simple terms, making it easy to understand, even for younger readers. We'll cover the structure, advantages, disadvantages, and provide relatable examples for each architecture.

2 Monolithic Architecture

2.1 Overview

Monolithic architecture is a traditional software design model where all components of an application are integrated into a single unit. This architecture is straightforward and often used in smaller applications where everything is developed and deployed together.

2.2 Structure and Design

In a monolithic architecture, the entire application is built as a single, unified codebase. All functions, such as UI, business logic, and data access, are bundled together in one package.

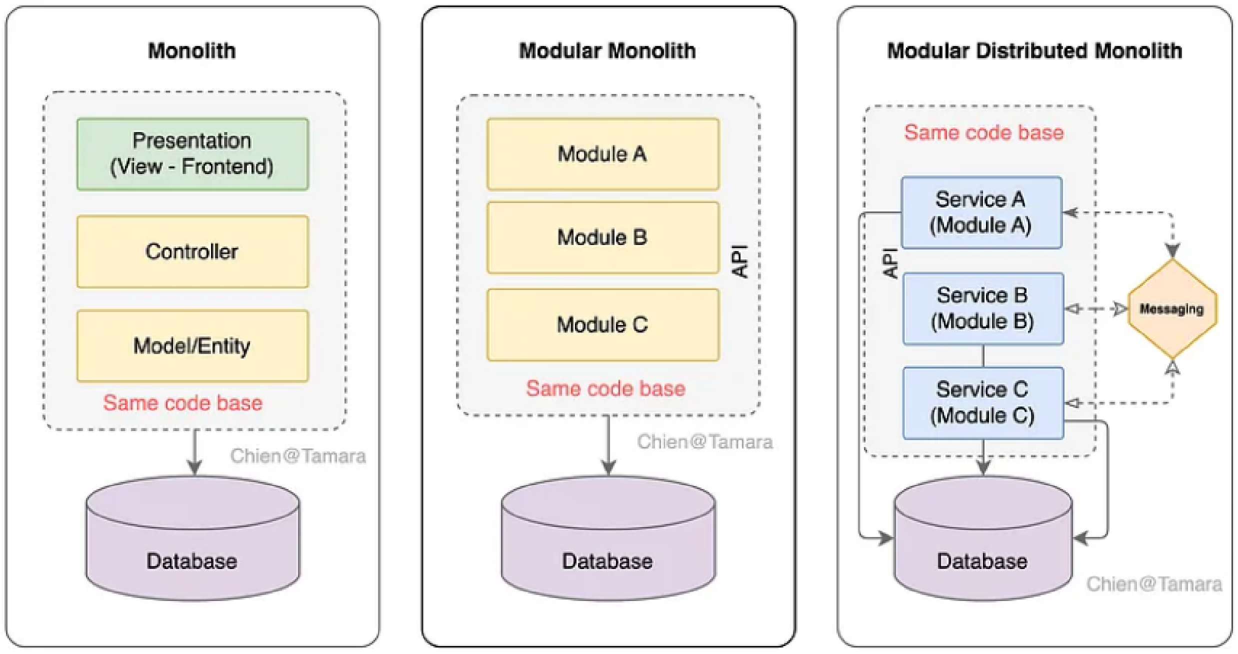

Figure 1: Monolithic architecture by Chien Hoang

2.3 Non-Functional Requirements Supported

- **Performance:** Since everything is in one place, the performance can be optimized easily.
- **Simplicity:** Easy to develop and deploy for small applications.

2.4 Advantages

- Simple to develop and deploy.
- Performance can be easily optimized.
- Easier to understand and maintain for small teams.

2.5 Disadvantages

- Not scalable: Difficult to manage as the application grows.
- Tight coupling: Changes in one part of the system can impact the entire application.

2.6 Example Scenario

A small blog website where the content, user interface, and database interactions are all managed within a single application.

3 Layered Architecture

3.1 Overview

Layered architecture divides an application into different layers, each with a specific responsibility. This separation makes it easier to manage and scale the application.

3.2 Structure and Design

The application is divided into layers such as presentation, business logic, data access, and database. Each layer only interacts with the layer directly below or above it.

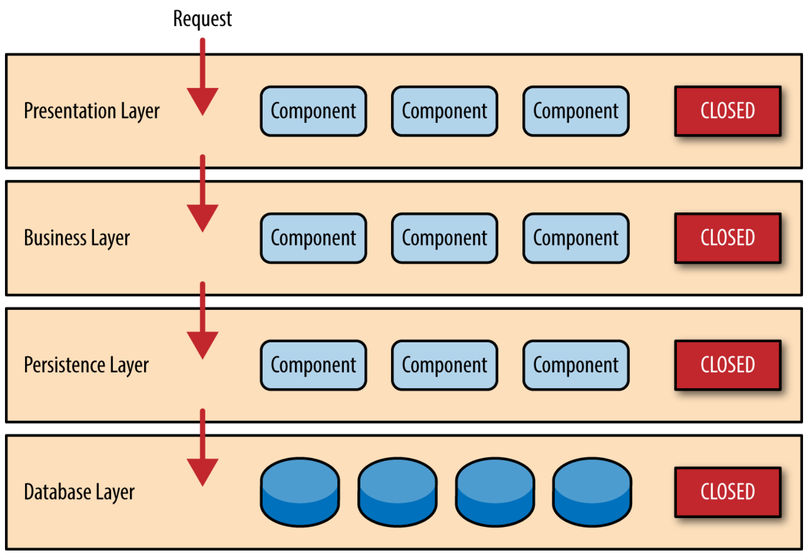

Figure 2: Layered architecture by O'Reilly Media

3.3 Non-Functional Requirements Supported

- **Modularity:** Enhances maintainability by separating concerns.
- **Scalability:** Easier to scale individual layers as needed.

3.4 Advantages

- Separation of concerns.
- Easier to manage and modify specific layers without affecting others.
- Reusability of layers.

3.5 Disadvantages

- Performance overhead due to multiple layers.
- Can become complex as the number of layers increases.

3.6 Example Scenario

A shopping website where the user interface, payment processing, and database operations are handled in separate layers.

4 N-Tier Architecture

4.1 Overview

N-Tier architecture is an extension of the layered architecture, where the application is separated into distinct tiers, typically involving the client, server, and database tiers. This setup supports distributed computing and scalability.

4.2 Structure and Design

The architecture is divided into multiple tiers such as presentation, application, and data. Each tier can run on different machines, allowing for distributed deployment.

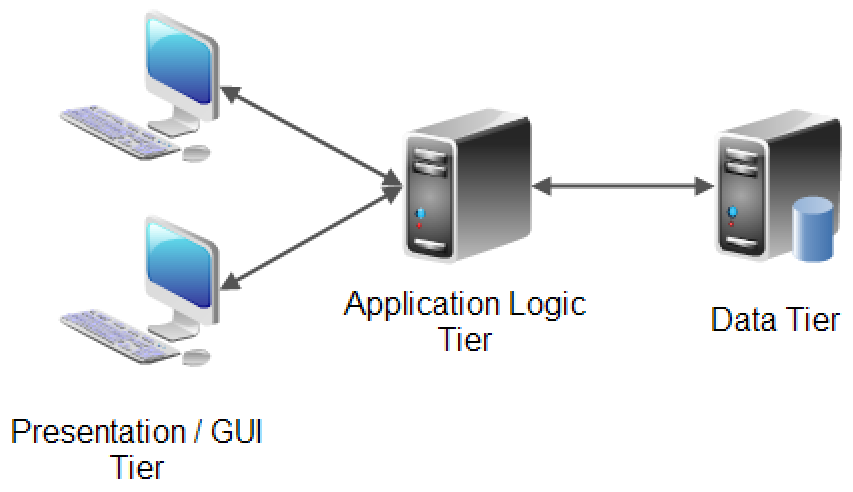

Figure 3: N-tier architecture by Jakob Jenkov

4.3 Non-Functional Requirements Supported

- **Scalability:** Tiers can be scaled independently.
- **Security:** Sensitive data can be isolated in specific tiers.

4.4 Advantages

- Supports distributed computing.
- Scalability across different servers or locations.
- Enhanced security by isolating layers.

4.5 Disadvantages

- Increased complexity in communication between tiers.
- Potential performance bottlenecks.

4.6 Example Scenario

An online banking system where the user interface, transaction processing, and database operations are handled on separate servers.

5 Service-Oriented Architecture (SOA)

5.1 Overview

Service-Oriented Architecture (SOA) organizes software components into reusable services that communicate over a network. Each service performs a specific business function and can be used across different applications.

5.2 Structure and Design

The architecture is built around services that interact with each other through a communication protocol like SOAP or REST. Each service is independent but can be composed to create more complex services.

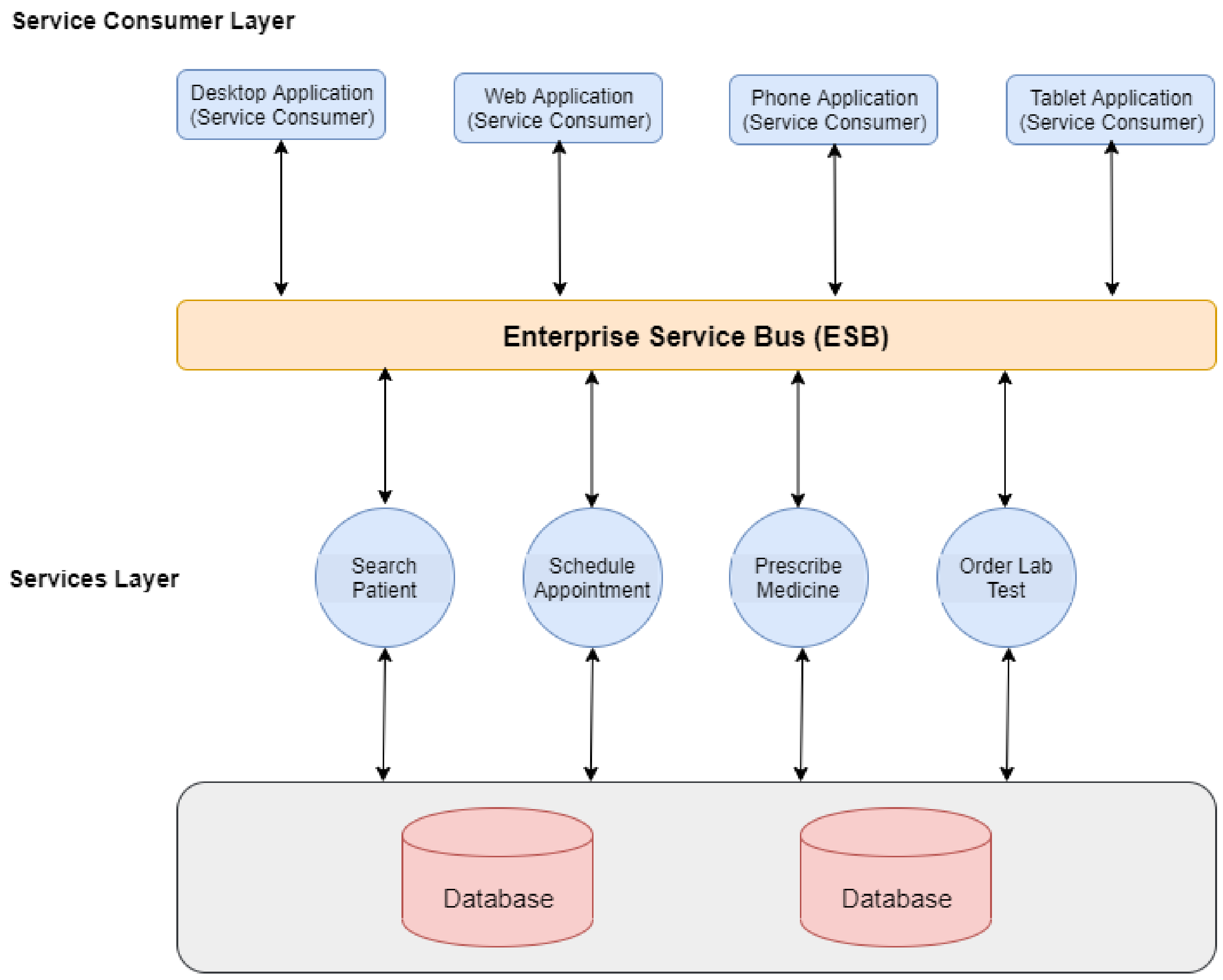
Service Consumer Layer
Desktop Application (Service Consumer)
Web Application (Service Consumer)
Phone Application (Service Consumer)
Tablet Application (Service Consumer)
Enterprise Service Bus (ESB)
Services Layer
Search Patient
Schedule Appointment
Prescribe Medicine
Order Lab Test
Database
Database

Figure 4: SOA by Adeel Sarwar

5.3 Non-Functional Requirements Supported

- **Reusability:** Services can be reused across multiple applications.
- **Interoperability:** Supports communication between different platforms and technologies.

5.4 Advantages

- High reusability of services.
- Facilitates integration with external systems.
- Platform-independent services.

5.5 Disadvantages

- High overhead due to communication protocols.
- Complex to manage and secure.

5.6 Example Scenario

A hospital management system where different services handle searching patient records, scheduling appointments, ordering of labs and medication prescription

6 Microservice Architecture

6.1 Overview

Microservice architecture breaks down a large application into smaller, independent services that communicate through APIs. Each service is responsible for a specific functionality, making the system more modular and flexible.

6.2 Structure and Design

The application is split into multiple small services, each with its own database and business logic. These services interact with each other using lightweight protocols like HTTP.

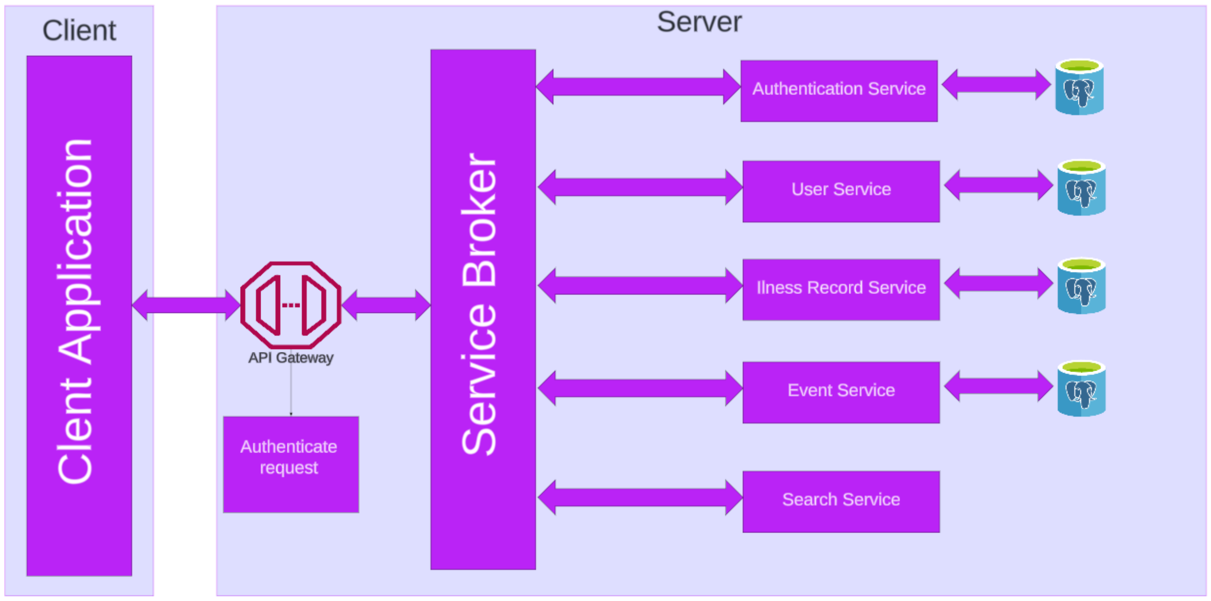

Figure 5: Microservice architecture of health record application by Ngane Emmanuel

6.3 Non-Functional Requirements Supported

- **Scalability:** Services can be scaled independently.
- **Resilience:** Failure in one service doesn't bring down the entire system.

6.4 Advantages

- Independent development and deployment of services.
- Enhanced fault tolerance and resilience.
- Easier to scale specific parts of the application.

6.5 Disadvantages

- Complex to manage multiple services.
- Increased need for monitoring and debugging tools.
- Requires robust inter-service communication.

6.6 Example Scenario

A health record manager application where different service handle, search, event recording, and user authentication.

7 Microkernel Architecture

7.1 Overview

Microkernel architecture, also known as the plug-in architecture, is designed around a core system that provides minimal functionality, with additional features implemented as plugins. This allows for flexibility and extensibility.

7.2 Structure and Design

The core system (microkernel) handles basic operations like process management, while additional functionalities are added through plugins that communicate with the core.

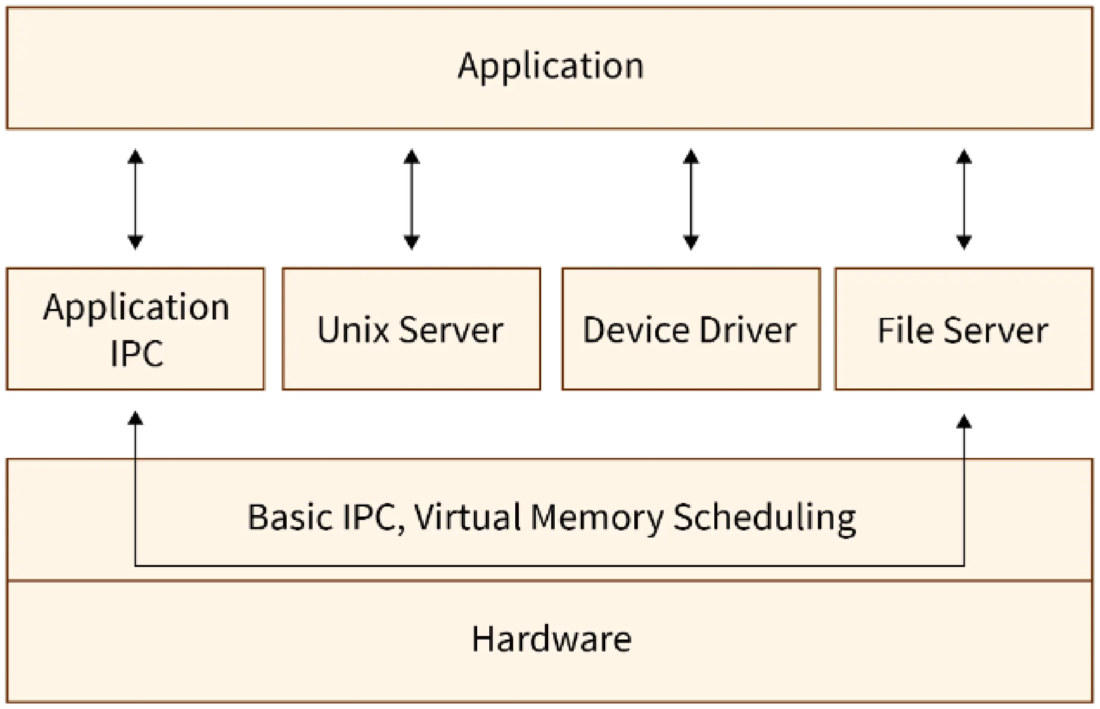

Figure 6: Microkernel architecture by Mansi

7.3 Non-Functional Requirements Supported

- **Extensibility:** Easily add new features through plugins.
- **Flexibility:** Modify or remove features without affecting the core system.

7.4 Advantages

- Highly modular and flexible.
- Core system remains lightweight.
- Easy to extend and customize.

7.5 Disadvantages

- Complex to manage dependencies between plugins.
- Performance overhead due to plugin communication.

7.6 Example Scenario

An operating system where the core handles essential tasks, and additional functionalities like file management or device drivers are added as plugins.

8 Event-Driven Architecture

8.1 Overview

Event-Driven Architecture (EDA) is based on the production, detection, and consumption of events. It is commonly used in systems where events trigger responses in real-time, making it ideal for highly dynamic environments.

8.2 Structure and Design

The architecture is divided into event producers, event processors, and event consumers. When an event occurs, it is passed through a processor, which triggers actions in the consumers.

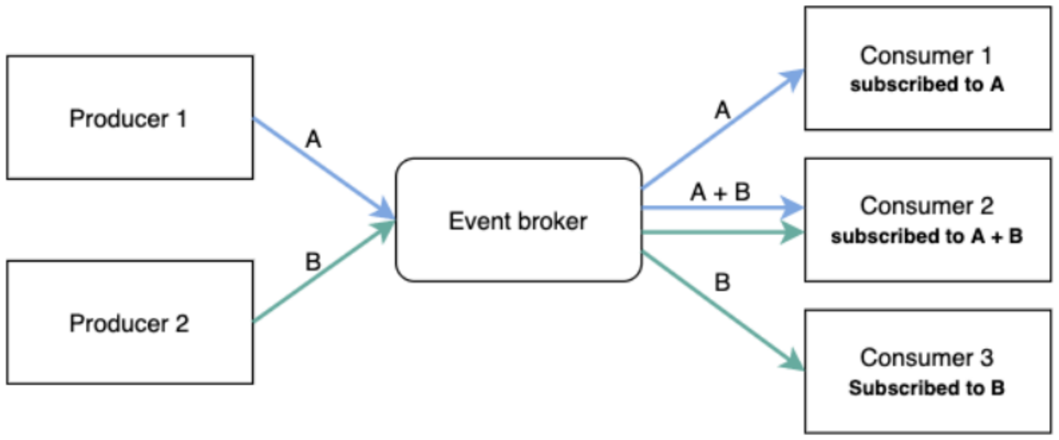

Figure 7: Event driven architecture by Stephen Roddewig via hobspot

8.3 Non-Functional Requirements Supported

- **Real-Time Processing:** Supports real-time response to events.
- **Scalability:** Event processors can be scaled independently.

8.4 Advantages

- Real-time event handling.
- Decoupled components improve flexibility.
- High scalability for processing events.

8.5 Disadvantages

- Complex to manage event flow and dependencies.
- Requires robust monitoring and error handling.

8.6 Example Scenario

A stock trading system where buy/sell orders trigger events that are processed in real-time to update user portfolios and market data.

9 Conclusion

In software development, one needs to know what the system should look like before developing. Being able to chose the right architecture for your system, can save you future cost and make the entire development and maintenance processes a ton easier. This handbook has covered all the essentials, giving you a clear idea on different architectures, their advantages, limitations, and the non-functional requirement each brings to the table. This should help you make better choices in developing software.

Happy Building!!!