S6 Portfolio Reading Guide

Jordy Walraven

Version history:

Version	Author(s)	Description	Date
1	Jordy Walraven	Setup first research question	28-3-2024

Distribution history:

Version	Distributed to	Comments	Date
1	Fontys Teachers	First portfolio delivery	14-4-2024

Table of Contents

Introduction

Problem

The sheer number of design and architecture patterns available is overwhelming. Each seems to offer a perfect solution, but selecting the right one for a system designed to handle a million users feels like navigating a maze. The core challenge lies in striking a delicate balance my non-functional requirements..

Through research, I aim to identify the ideal design patterns and architectural approaches that can overcome these challenges. This will help me build a system that is not only scalable but also remains understandable and manageable for future development and maintenance.

Research questions

Main Research: How to design and deploy a web application that enables user-submitted system performance evaluation for game compatibility, ensuring scalability, security, reliability, maintainability, and high performance?

Sub Questions:

- What architectural styles best suit the requirements of my system?
- How do different services or components communicate with each other?
- How can I ensure that the application is secure?
- How to deploy a scalable application?
- How do I make sure the application is reliable deployed without breaking issues?
- How do I make sure the application is reliable running in the cloud?

What architectural patterns best suit the requirements of my system?

There are a lot of different architectural styles you need to consider when creating your own web application. Especially when looking at requirements like scalability, security, reliability, maintainability and performance. First we will have a look at the different architecture styles that exist, and we will compare these with each other to see which best suit the requirements of the system.

Which architecture styles exist and what are some pros and cons?

There exists an abundance of architectural styles, making it impractical to delve deeply into each one. Therefore, I will focus on the 13 most widely utilized patterns, as they possess sufficient support and documentation. Other styles, lacking in such resources, will not be explored in this context. The primary source I will be using is an article by (Ritvik Gupta, 2023).

The patterns I will be researching are:

- Layered pattern
- Client-server pattern
- Event-Driven pattern
- Microkernel pattern
- Microservice pattern
- Broker pattern
- Event-bus pattern
- Pipe filter patten
- Blackboard pattern
- Component-based pattern
- Service-oriented architecture
- Monolithic architecture
- Space-based architecture

Layered Pattern

The layered pattern (Kaseb, 2022) is a pattern where each responsibility of the application is separated in layers. For example an API might consist of a presentation layer (controllers), Business layer and data access layer. This modular approach promotes segregation of concerns and facilitates easy maintenance and scalability in a complex system.

Pros	Cons
Promotes separation of concerns	Overhead
Encourages modularity	
Supports parallel development	

Client-Server pattern

The Client-Server pattern (Gayantha, 2020) is a fundamental architectural design that divides an application into two distinct components: the client and the server. This separation enables efficient communication and enhances scalability and maintainability. In this pattern, the client is typically a user interface or application that interacts directly with the end user, while the server is responsible for managing data processing, storage, and business logic.

Pros	Cons
Scalability	Data manipulation
Security	
Reliability	

Event-Driven pattern

The Event-Driven pattern enables systems to respond dynamically to events, such as user interactions or data updates. Events are emitted by producers and consumed by handlers, allowing for asynchronous, loosely-coupled communication. This pattern fosters flexibility, scalability, and real-time responsiveness in software architectures.

Pros	Cons
Asynchronous	Complexity
Scalable	Debugging
Modular	Overhead
Scalable	

Microkernel pattern

The microkernel architecture pattern (Heusser, 2020) is a software design approach that sits between monolithic and microservices architectures. It is characterized by a central core system that contains only the essential components necessary to run the application, and additional functionalities are implemented as plugins. This architecture allows for a high degree of modularity and flexibility, as new features or modifications can be added or changed without altering the core system. The core system and plugins communicate through interprocess communication mechanisms provided by the microkernel, ensuring that they remain isolated from each other.

Pros	Cons
Modularity	Performance overhead
Flexibility	Complexity
Stability	Dependent on kernel

Microservices

Microservices architecture (IBM, n.d.) is a cloud-native approach where a single application is composed of many loosely coupled and independently deployable smaller components or services. These services are organized by business capability, often referred to as a bounded context. This architecture allows for easier code updates, as new features or functionality can be added without affecting the entire application. Teams can use different stacks and programming languages for different components, and components can be scaled independently, reducing waste and cost associated with scaling entire applications.

Pros	**Cons**
Fault isolation	Complexity
Independent scaling	Resource intensive
Smaller and faster deployment	Data inconsistency
Scalability	

Broker pattern

The Broker Pattern (Wikipedia, 2023) is an architectural pattern used to structure distributed software systems with decoupled components that interact by remote procedure calls. It involves an intermediary software entity, known as a broker, which is responsible for coordinating communication between clients and servers. The broker acts as a middleman, receiving messages from one component and forwarding them to the appropriate recipient. This pattern allows components to remain decoupled and focused on their own responsibilities, while still being able to communicate and collaborate with other components in the system.

Pros	**Cons**
Decoupling	Single point of failure
Scalability	Complexity
Changeability	Performance overhead

Event-Bus pattern

The Event Bus pattern is a design pattern that facilitates communication between components in a distributed system by using an event bus as a central hub. This pattern is particularly useful in large-scale applications where components need to interact without being tightly coupled, adhering to principles of loose coupling and separation of concerns. The event bus acts as a pipeline, where components (referred to as subscribers) can register to receive specific types of events. When an event occurs, it is dispatched to the event bus, which then forwards it to all registered subscribers that are interested in that type of event. This mechanism allows components to communicate asynchronously, enabling them to operate independently and react to events as they occur.

Pros	**Cons**
Loose coupling	Complexity
Asynchronous	Debugging
Scalability	Performance overhead

Pipe-filter pattern

The Pipe and Filter pattern is an architectural pattern that structures a system as a sequence of processing elements, where each element is a filter that performs a specific operation on the data. Data flows through the system in a pipeline, where each filter takes input from the previous filter and passes its output to the next filter in the sequence. This pattern is particularly useful for data processing and transformation tasks, where operations can be performed in a series of steps.

Pros	**Cons**
Modularity	Complexity
Flexibility	Latency

Blackboard pattern

The Blackboard pattern is a way to solve complex problems by breaking them down into smaller, manageable parts. It involves a central "blackboard" where different parts of a program, called "agents," can share information and work together to find a solution.

Pros	**Cons**
Modularity	Complexity
Flexibility	Coordination
Collaboration	Security

Component-based pattern

Component-based architecture is a software design approach that structures applications into reusable, modular components. Each component encapsulates specific functionality and communicates with others through well-defined interfaces, promoting decoupling and flexibility. This architecture enhances development efficiency, reliability, and scalability by facilitating the reuse of components across different parts of an application or even in different applications. It supports the creation of extensible, encapsulated, and independent components that can be easily replaced or extended without significant disruption to the overall system. Despite its advantages, component-based architecture may not be suitable for every scenario, especially when applications are large or when the need for customization limits the reusability of components.

Pros	**Cons**
Modularity	Performance issues
Easy maintenance	
Easy testing	

Service-oriented architecture

Service-Oriented Architecture (SOA) is an approach to designing software systems where different functions or services are organized to communicate and work together effectively. In SOA, services are like building blocks, each performing specific tasks and capable of independent deployment. These services encapsulate multiple functionalities within a specific domain. SOA promotes flexibility, scalability, and interoperability by breaking down complex systems into manageable parts, though the services are typically larger and encompass more functionalities compared to microservices.

Pros	**Cons**
Reusability	Complexity
Easy maintenance	
Flexibility	

Monolithic architecture

Monolithic architecture refers to a traditional approach to software design where all components of an application are tightly integrated into a single, indivisible unit. In this model, the entire application, including its user interface, business logic, and data access layers, is developed and deployed as a single unit.

Pros	**Cons**
Simplicity	Scalability
Easy of development	Maintainability
	Deployment

Space-based architecture

Space-based architecture is an architectural pattern where data is distributed across a grid of interconnected nodes, often referred to as a "space." This architecture is characterized by its ability to scale horizontally and handle large volumes of data by partitioning and replicating data across multiple nodes. It allows for high availability, fault tolerance, and scalability, making it suitable for applications with demanding performance requirements. Space-based architectures are commonly used in distributed caching, data grids, and real-time analytics systems.

Pros	**Cons**
Scalability	Complexity
Performance	Data Consistency
	Resource Overhead

Which architecture patterns are most suitable for a web application that fits the requirements?

Now that we have taken a look at which styles there are, we need to decide which ones fit our requirements best. We will be making a highly scalable, maintainable, reliable, performant and secure application. Keeping this in mind we will choose which patterns to use, so down below there is a table with all above patterns and if they are applicable to my application.

Pattern	**Applicable**	**Description**
Microservice pattern	Yes	This pattern is central to my requirements. It decomposes CIRI2 into smaller loosely coupled services, each responsible for a specific business function. This allows for independent scaling,
Layered pattern	Yes	This pattern can be applied within each microservice to organize its internal architecture, separating concerns such as presentation, business logic and data access. It aids maintainability.
Client-Server pattern	Yes	This pattern is implicit in my architecture, as I will be using a SPA that serves as client side, and my microservices that act as a server side. It ensures clear separation of concerns and support scalability
Event-Driven pattern	Yes	Using this pattern for communication between my microservices would be beneficial for ensuring loose coupling and scalability. Each microservice will then communicate asynchronously with each other.
Broker pattern	Yes	For communication between my microservices a broker pattern would be ideal, as this helps decoupling and scalability.
Component-based pattern	Yes	Because I will be using a SPA my frontend needs to stay modular and use reusable component, this can help with maintainability and extensibility
Service-oriented architecture	Yes	While microservices are a form of SOA, explicitly following SOA principles can help in designing services that are loosely coupled, reusable, and interoperable. It emphasizes service abstraction, standard interfaces, and service composition, which align with the goals of scalability, reliability, and maintainability.
Space-based architecture	Maybe	Space-based architecture is an alternative to traditional microservices architecture that focuses on distributing data and processing across multiple nodes in a shared memory space. Unlike microservices, which involve separate services communicating via APIs, space-based architecture uses a centralized space for communication and coordination.
Micro-Kernel pattern	No	Because the application needs to be scalable and reliable, I don't want a central kernel. I want everything to be completely standalone, this pattern might fit better for a operating system.
Pipe-filter pattern	No	Pipe filter pattern doesn't really address the requirements of the system. This is because I won't be doing that much processing on my data.

BlackBoard-Pattern	No	Blackboard pattern has a centralized nature, which will introduce scalability and reliability challenges. It also holds shared domain knowledge, when you want to keep this separated.
Monolithic architecture	No	Monolithic are a centralized way to handle you application, this absolutely does not fit the goals I am trying to meet. Monolithic application have challenges in scalability agility, and flexibility. And also comes with other challenges, like big deployments etc.

Bibliografie

(2023, September 7). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Broker_pattern#:~:text=The%20broker%20pattern%20is%20an,for%20transmitting%20results%20and%20exceptions.

Gayantha, K. (2020, Januari 2020). *Client-Server architectural pattern*. Retrieved from Medium: https://medium.com/@96kavindugayantha/software-architectural-patterns-17b05bfa3aef

Heusser, M. (2020, March 10). *What is a microkernel architecture, and is it right for you?* Retrieved from TechTarget: https://www.techtarget.com/searchapparchitecture/tip/What-is-a-microkernel-architecture-and-is-it-right-for-you

IBM. (n.d.). *What are microservices?* Retrieved from IBM: https://www.ibm.com/topics/microservices

Kaseb, K. (2022, March 27). *The Layered Architecture Pattern in Software Architecture*. Retrieved from Medium: https://medium.com/kayvan-kaseb/the-layered-architecture-pattern-in-software-architecture-324922d381ad

Ritvik Gupta, A. S. (2023, October 15). *software-architecture-patterns-types*. Retrieved from Turing: https://www.turing.com/blog/software-architecture-patterns-types/