Research Report

Jordy Walraven

Table of Contents

[Introduction 1](#_Toc162251309)

[Problem 1](#_Toc162251310)

[Research questions 2](#_Toc162251311)

[What architectural styles best suit the requirements of my system? 2](#_Toc162251312)

[Which architecture styles exist and what are some pros and cons? 2](#_Toc162251313)

# 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 styles 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 14 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).

**Layered Pattern**

|  |  |  |  |
| --- | --- | --- | --- |
| Architectural style | Description | Pros | Cons |
| Layered Pattern | The Layered Pattern, also known as the Layered Architecture, organizes software into distinct layers, each responsible for specific tasks. This modular approach promotes separation of concerns and facilitates maintenance and scalability in complex systems. | Promotes separation of concerns  Encourages modularity  Supports parallel development | Overhead of communication |
| Client-Server Pattern | The client-server pattern is a model where clients request services or resources from servers, which provide those services or resources. Communication typically occurs over a network, with clients initiating requests and servers responding accordingly, facilitating distributed computing. | Security  Scalabilty  Shared resources | Data manipulation |
| Event-Driven Pattern | The event-driven pattern is a programming paradigm where the flow of the program is determined by events occurring in the system. Instead of following a linear sequence of execution, the program responds to events triggered by user actions, system notifications, or other sources, allowing for asynchronous and reactive behavior. | Asynchronous  Scalable  Modular | Complexity  Debugging  Overhead |
| Microkernel Pattern | The microkernel pattern is an architectural design where a minimalistic kernel provides basic services, and additional functionality is implemented as separate modules. This approach promotes flexibility, modularity, and extensibility, but it may incur performance overhead due to communication between modules. | Modularity  Flexibility  Extensibility | Performance  Complexity  Scalability |
| Microservices | Microservices is an architectural approach where software is composed of small, independently deployable services that communicate over well-defined APIs. This enables agility, scalability, and allows for easier maintenance and updates, but can introduce complexity in managing distributed systems. | Scalability  Flexibility  Modularity  Resilience | Complexity  Overhead  Deployment |
| Broker pattern | The Broker Pattern is a design pattern used in distributed systems where a broker acts as an intermediary between clients and servers, facilitating communication and coordination. Clients interact with the broker to request services or resources, and the broker routes these requests to appropriate servers, decoupling the clients from specific server implementations. | Decoupling  Scalability  Flexibility  Reliability | Complexity  Single point of failure |
| Event-bus-pattern | The Event-Bus Pattern is a design pattern where components within a system communicate through a central event bus rather than directly with each other. Components can publish events to the bus, and other components can subscribe to receive and react to those events, promoting loose coupling and modularity within the system. | Loose coupling  Scalability | Complexity    Debugging |
| Pipe-Filter Pattern | The Pipe-Filter Pattern is a design pattern where data processing tasks are divided into smaller, independent components called filters, which are connected in a pipeline. Each filter performs a specific operation on the data and passes it along to the next filter, enabling modular, reusable, and scalable data processing workflows. | Modularity  Reusability  Scalability | Complexity  Overhead |

# Verwijzingen

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