Architecture research document

for

HeardIT

Author: Mihail Vasilev

Date: 13-04-2024

Contents

[1. Introduction 3](#_Toc169364461)

[2. Main question 3](#_Toc169364462)

[3. Sub-questions 3](#_Toc169364463)

[4. Conclusion 8](#_Toc169364464)

[5. Architecture design 9](#_Toc169364465)

1. Introduction

The purpose of this document is to outline the research process taken during the architecture design process of the HeardIT application. The research is split into the following sections *– Main research question*, where the main question is defined, *Sub-questions*, where the sub-question derived from the main questions are defined and answered and *Conclusion*, where the answers to the sub-questions are combined in order to answer the main question. The final section is the *References* section where the sources of the information used during the research are presented.

1. Main question

In this section the main question will be established. In order to complete the research, the main question needs to have a concrete answer. For this reason, it is important to define the main question well. The main question will also allow us to create the sub-questions that will help us answer the main question.

The main research question is:

*What is the most suitable architecture type for the HeardIT web-application?*

Answering this question will allow us to determine the best architecture design for the HeardIT application. In the next section I will establish the sub-questions that were derived from the main question and I will formulate an answer to each of them.

1. Sub-questions
2. What type of architecture designs are suitable for the HeardIT application?

Choosing the right architecture for a music streaming web-application can be a difficult and complex process. There are many different pre-established architecture design patterns that can be used as the basis of the application. For this reason, I am going to look over my requirements and depending on them make a decision.

Let’s establish first what are the main and most widely used design patterns. According to Wayner (2021) and GfG (2021) the main software architecture designs that are most widely used for developing are the following: *Layered architecture, Event-driven architecture, Microkernel architecture, Microservices architecture* and *Space-based architecture*. Some of these might suit my requirements for implementing HeardIT but first I am going to explain each of them and dive into more detail. (Wayner, 2021)

Here is a summary of the main capabilities of each of these architecture design patterns:

1. Layered architecture – software is organized into distinct layers – presentation, business logic, persistence, each layer represents a specific responsibility and establishes separation of concern and modular design. Layers can communicate only with their adjacent layers and follow a well-structured and easy to understand, test and implement system.

As an example, a banking application that has - presentation layers for customer interfaces, business logic layers for transaction processing and data access layers for interacting with databases. (GeeksforGeeks, 2021)

1. Event-driven architecture – decouples components and relies on events to trigger actions. This enables asynchronous communication and scalability. Events are used only by the components that need them which allows them to remain flexible. This pattern is suitable for systems with unpredictable workloads, however significant complexities can arise in the managing of the event flows and this can lead to loss of reliability. (Wayner, 2021)

As an example, a smart home automation system might utilize event-driven architecture to respond to sensor data, triggering actions like adjusting temperature or turning on lights.

1. Microkernel architecture – the architecture is split into a core set of essential functions and optional modules. The core provides the basic services, while modules add specific functionalities and customizations to the system. This pattern can be easy to maintain and upgrade but may introduce complex issues related to the communication between the modules. (Wayner, 2021)

As an example, the Eclipse IDE follows a microkernel architecture where core functionalities are kept minimal and additional features are added via plugins.

1. Microservices architecture – the application is structured as a collection of loosely coupled, independently deployable services, each responsible for a specific capability. Services communicate through APIs. This pattern promotes flexibility, scalability and continues delivery. It also promotes collaborative environment where multiple teams can be working on different services within the same bigger application. It fosters rapid development and fault isolation and reliability. Careful management of service boundaries is required and operation complexity can be also very high. (GeeksforGeeks, 2021)

As an example, Netflix, Spotify and many other similar mainstream applications use microservices to handle user account management, content delivery, and recommendation algorithms.

1. Space-based architecture - different components are decoupled through the use of a shared communication space, where the data is published and distributed to the correct component. Components interact indirectly through the shared space promoting loose coupling and flexibility. Distribution systems with high concurrency and big difference between the workloads benefit the most from this kind of architecture. (Wayner, 2021)

As an example, financial trading systems utilize space-based architecture to distribute and process incoming market data across multiple nodes, allowing for high throughput and fault tolerance in handling trading decisions.

Now that the main architecture design patterns are established, I wanted to narrow down my choice so that I could create a more detailed overview and have better insight into which of these would be most suitable for my application.

After careful consideration, I narrowed my options to the following two design patterns: *Layered architecture* and *Microservices* architecture designs. I am going to compare them and make a decision based on my requirements. I have chosen to go more in-depth into these two because for music streaming application such as HeardIT I see these two architecture design patterns as the most suitable ones.

In the following sections I will explore more specific contexts for each architecture design pattern.

Methods used:

* Literature study – this method was used to determine the main architecture design patterns and their use cases and specifications
* Available product analysis – this method was used when researching what design patterns are used and for what kinds of applications they are suitable
* Design-pattern analysis – this method was used to evaluate the different ways I can establish my architecture design

1. Which architecture design meets the scalability and stability requirements?

Now that I have narrowed down my choice for architecture design pattern to either the Layered design or the Microservices, I wanted to compare them side by side and determine which of the two would be more suitable for my application.

In this section I will focus on the scalability and stability that each of them provides in order to see which pattern would allow me to meet the criteria for my kind of application. (Atlassian, n.d.)

|  |  |  |
| --- | --- | --- |
| Design Pattern | Pros | Cons |
| Layered architecture | Simpler development process, deployment, and testing. One codebase that has to be deployed as one whole piece. Due to this, it might be easier for developers to understand the whole picture and have a good idea of how each part of the application works. Unified programming language and environment. | Challenging in terms of scalability since the entire application has to be scaled together. Can lead to inefficient use of resources since some parts of the application might not require scaling while others require it.  Stability can be compromised due to its closely connected parts. A bug or an issue can bring the whole system down, affecting all users and functionalities. |
| Microservices | Designed with scalability in mind since all individual services can be scaled independently based on demand. Services under high loads can be scaled up while others remain unaffected, which promotes consistent resource utilization.  Services are isolated from each-other and failure of one service does not affect the overall system, but only the specific functionality. | Complex communications between the services. Many different components which in some cases might even be implemented in different programming language and environments. This can lead to difficulties when trying to pinpoint a specific issue or failure. |

From this comparison, it can be clearly seen that the *Microservices* architecture design pattern clearly is more suitable for the HeardIT application. My application is going to consist of many different services and components that have to be scalable under high loads while also keeping high application stability and availability.

Methods used:

* Literature study – this method was used to determine the main advantage and disadvantages of each of the architecture design patterns
* IT architecture sketching – this method was used to evaluate which architecture design would be most suitable to my requirements

1. Which architecture design meets the maintainability requirements?

Continuing my comparison between the Layered design and the Microservices, this time I am going to compare them side by side and determine which of the two has an easier and more maintainable structure. This has to be taken into account since the HeardIT application will be developed and maintained by using the Agile methodology. This means that updates and fixes will be frequently introduced to the application throughout its lifecycle. (Atlassian, n.d.)

|  |  |  |
| --- | --- | --- |
| Design Pattern | Pros | Cons |
| Layered architecture | Development process can be streamlined due to the shared codebase.  Deployment has to be done of the whole application instead of different components  Testing can be done to the whole application at once which simplifies the testing process. | Maintaining can become increasingly challenging as the application increases in size and complexity.  Updates to one part of the application may affect another part which can lead to unanticipated failures and higher risks or introducing bugs or issues. |
| Microservices | Each service tackles a specific functionality which means that development teams can focus on it without having to worry that other parts of the application will be directly affected by it.  Less unexpected dependencies since services in general are independent from one-another.  Services can be developed, tested and deployed individually reducing the risk and impact of changes. | Communication of the services can be complex and require multiple teams to get involved when trying to fix a bug or an issue.  Failures can be harder to localize due to the complex connections between the different services. |

From this comparison, it can be clearly seen that the *Microservices* architecture design pattern is more suitable for the HeardIT application in terms of maintainability. Different services can be developed without affecting the existing functionalities and without compromising availability.

Methods used:

* Literature study – this method was used to determine the maintainability aspects of the two architecture patterns
* IT architecture sketching – this method was used to evaluate which architecture design would be most suitable to my requirements

1. Which architecture design meets the requirements for cloud native applications?

The next part of my comparison between the Layered design and the Microservices is to determine which of the two has better cloud native capabilities. This has to be taken into account since the HeardIT application will be deployed to the cloud and will have to be suitable to the modern cloud environments. (Atlassian, n.d.)

|  |  |  |
| --- | --- | --- |
| Design Pattern | Pros | Cons |
| Layered architecture | Deployment has to be done of the application as a whole. One thing to deploy and manage. | Due to the fact that the whole application has to be deployed as one whole unit, this can lead to not taking full advantage of the cloud capabilities.  This type of self-contained architecture can limit the ability to scale and can lead to potential waste of resources. |
| Microservices | Inherently well suited for cloud integration due to its nature of different and independent services.  Each service can be containerized using technologies like Docker and orchestrated using Kubernetes.  Promotes principles such as scalability and stability which align closely with the cloud-native best practices. (Microservice Architecture Pattern, n.d.) | Numerous services running independently can lead to difficulties in coordination in managing service discovery, deployments and inter-service communication can become challenging especially at scale. |

Once again, it can be clearly seen that the *Microservices* architecture design pattern is more suitable for the HeardIT application in terms of cloud native application requirements. Services can be scaled and deployed independently and with the proper orchestration and communication tools, a web-application such as HeardIT can become much more consistent in its performance.

Methods used:

* Literature study – this method was used to determine the cloud compatibility aspects of the two architecture patterns
* IT architecture sketching – this method was used to evaluate which architecture design would be most suitable to my requirements
* Community research – this method was used to determine what are the benefits of each architecture design in the context of the cloud native sphere

1. Which architecture design meets the security requirements?

The final part of my comparison between the Layered design and the Microservices is to determine which of the two has the better security by design and can meet the requirements for security and data privacy. This has to be taken into account since the HeardIT application will be used by many users and the data that is circulating the application services must be securely and reliably stored and used. (Atlassian, n.d.)

|  |  |  |
| --- | --- | --- |
| Design Pattern | Pros | Cons |
| Layered architecture | Security measures are typically implemented at the application level.  Security controls and mechanisms are applied uniformly throughout the whole application. | Can be challenging to manage and maintain due to its nature to be applied across the entire application.  Breaches in one part of the application can potentially compromise the security of the complete application. |
| Microservices | Each service has its own security measures and controls tailored to its specific functionality and data requirements.  Reduced “attack surface” due to its isolation from the other services and containment scope in the event of a security breach.  Security measure can be scaled accordingly with the individual microservices ensuring that the data protection remains robust at all times. (Microservice Architecture Pattern, n.d.) | Implementation of consistent security measures across a large number of microservices can lead to challenging situations and additional tools, processes, or expertise.  Ensuring proper encrypted communication, access controls, and data protection mechanisms across various services adds to the complexity of the application which can potentially increase the risk of misconfiguration or oversight, leading to security vulnerabilities if not managed effectively. |

From this comparison, it can be once again seen that the *Microservices* architecture design pattern is more suitable for the HeardIT application in terms of security requirements. Due to the nature of the application, personal data has to be managed carefully and protected from potential threats. For this reason, the Microservices architecture design is more suitable since it allows me to implement suitable to each functionality protection measures while also minimizing the “attack surface” that can be used to cause harm to the users.

Methods used:

* Literature study – this method was used to determine the security benefits and downfalls of the two architecture patterns
* IT architecture sketching – this method was used to evaluate which architecture design would be most suitable to my requirements
* Security test – this method was used to determine the security aspects of each of the discussed architecture design patterns

1. Conclusion

To conclude this research, I am going to design the HeardIT application architecture to follow the Microservices architecture design pattern to establish the architecture of the HeardIT application. After carefully considering my options, conducting extensive research into the possible architecture designs that I could follow, I determined that the Microservices design pattern is the most suitable one, due to its scalability, stability, maintainability, cloud native design and security by design aspects that it can provide me.

1. Architecture design

Now that the architecture design pattern has been established, I have come to the point where I can create the actual architecture of HeardIT. My application has functional requirements and non-functional requirements that need to be addressed in order for the application to be complete. Depending on the priority of these requirements, I have made a chart where I show at what stage of the development process each requirement will be addressed. For more specific information, refer to - *User Requirements - HeardIT.docx*. My architecture consists of several services, each of them specialized in a specific functionality of the application. I aggregated common functionalities that use the same resource into these services, based also on the predicted amount of load each functionality can be expected to have to handle. For example, in my current architecture, my *search\_service*, which is responsible for doing the searching of specific songs, is expected to be one of the most commonly used functionalities in HeardIT. As such, it is a separate service that can be scaled automatically and is not dependent on other services. It focuses on the searching for specific songs by multiple parameters and can be extended and further updated, without affecting other HeardIT components and services. This philosophy is demonstrated in the other services that HeardIT has.

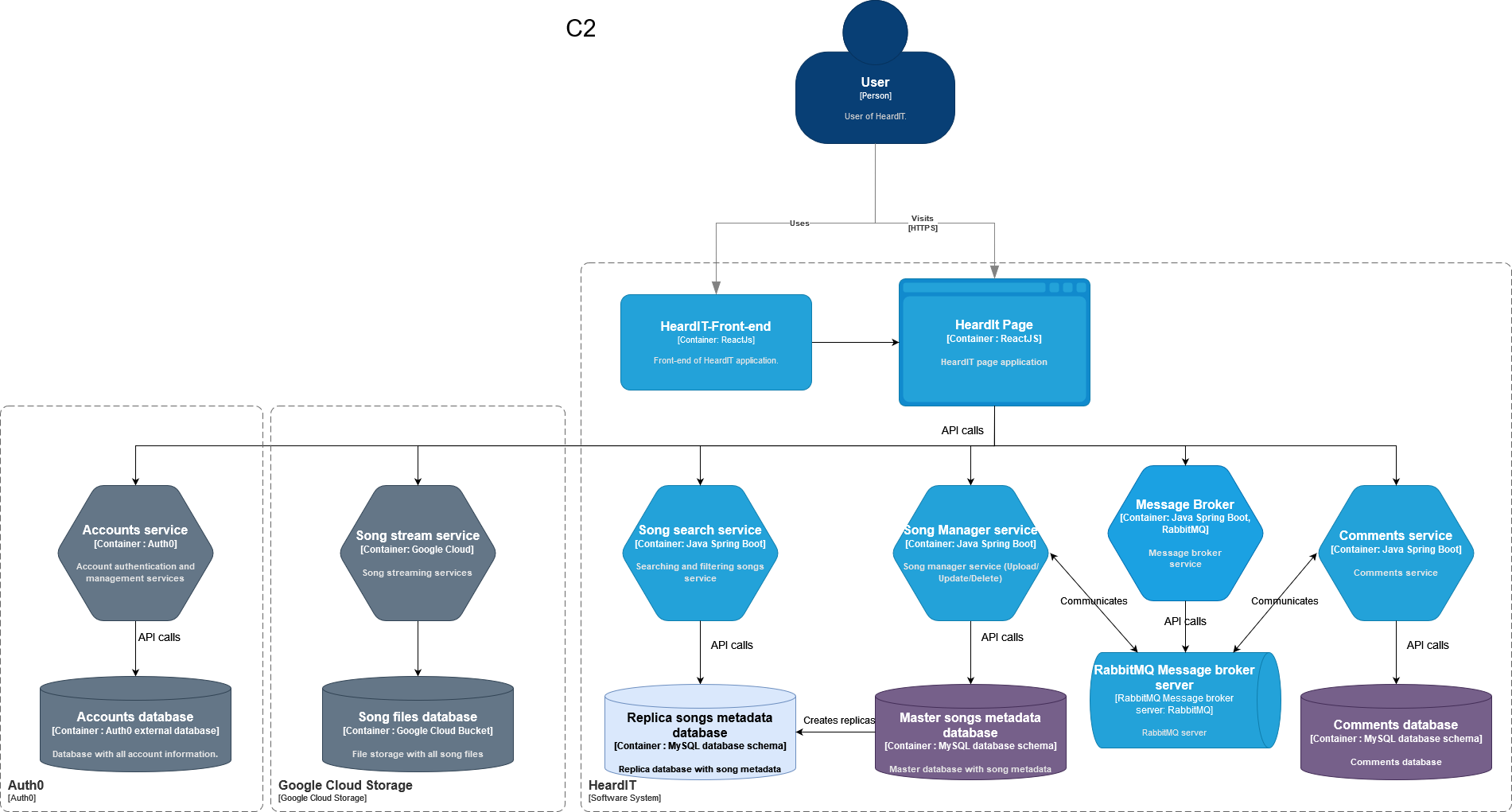
In Figure 1, I have described the rest of the services and their interactions in the form of a diagram.

Figure 1 - HeardIT architecture C2

This diagram represents the current architecture that has also been implemented. As such, it will be continuously updated. The main principles for microservice architecture are applied and will be applied for each of the next services that are implemented.

**References:**

* Wayner, P. (2021, June 19). How to choose the right software architecture: The top 5 patterns. TechBeacon. <https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choic>
* GfG. (2021, October 27). Types of software architecture patterns. GeeksforGeeks. <https://www.geeksforgeeks.org/types-of-software-architecture-patterns/>
* Atlassian. (n.d.). Microservices vs. monolithic architecture | Atlassian. <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a,monolith%20architecture%20for%20software%20design>.
* Microservice Architecture pattern. (n.d.). microservices.io. <https://microservices.io/patterns/microservices.html>