Data distribution

research document

for

HeardIT

Author: Mihail Vasilev

Date: 20-05-2024

Contents

[1. Introduction 3](#_Toc163978462)

[2. Main question 3](#_Toc163978463)

[3. Sub-questions 3](#_Toc163978464)

[4. Conclusion 8](#_Toc163978465)

1. Introduction

The purpose of this document is to outline the research that was undertaken to determine the most suitable strategies and processes for handling data by 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. This 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 approach for distributing and storing the data used by HeardIT?*

Answering this question will allow us to determine the best way to store and handle the required data 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 data is HeardIT working with?

The very first thing that needs to be established from the very beginning is what types of data HeardIT will have to utilize. HeardIT is a music streaming service that provides its users with tools to interact with the songs, artists and other users. It is an application that provides a more comprehensive way to experience music for its users providing different features that allow its users to not only listen but also get to learn the songs and communicate with the artists, or upload their own tracks and provide instructions and details on how to play them.

With this established, we can outline the three main subsets of data that has to be stored and handled:

* Songs data
* User data
* Extra supporting data

I decided to separate them in this way since these three main subsets can be aggregated into three different research sub-questions due to the complexities that are related to each.

Songs can be separated as their own special kind of data that has to be handled since they contain the most complex data type that has to be used: large files. These files can be of different sizes and types whose storing and handling becomes a difficult challenge that needs to be overcome to provide streaming to a large audience of users. This is one of the main obstacles that can determine whether the application is capable of meeting the requirements for enterprise level applications.

User data is just as important, since it can contain personal details that needs to be stored and handled safely and securely. This is another main challenge since in our modern internet environment is important more than ever to keep your users’ information secure while also not hampering the experience and allowing the users to have a pleasant time using the application. Moreover, improper storage and usage of user data can violate legal regulations and result in serious legal consequences. For this reason, user data handling and storing deserves its own discussion.

The last subset of data contains all of the supporting data will be uploaded and used on the web-application. This can contain data like lyrics, song note or tab sheets, user comments, pictures and other types of data that is not considered private or sensitive. I decided to aggregate these kinds of data into their own subset that can follow a more general type of storage and handling system.

With this I can conclude with determining the types of data HeardIT has to work with. In the following sections I will provide the research process and results that were created for each of the three main subsets of data.

Methods used:

* Domain modelling – this method was used when researching what kinds of data HeardIT will have to store and handle
* Problem analysis – this method was used to determine what kind of issues and problems can arise when working with the different kind of information

1. What is the most suitable approach for storing and handling the data for the songs?

Storing song data while also being easily accessible to HeardIT users is one of the main challenges. There are several types of files that can be used to record and keep music, like for example mp3 or wav. These files can also be large in size and contain a lot of data which the application needs to be able to stream in real time to the users. Since I have had little experience with handling and storing such kinds of data, I decided that the best course of action is to find out what other big song streaming services have done to tackle these issues.

The first two most popular music streaming web services that come to mind are Spotify and SoundCloud. These two music providers are renowned for their fast, stable and clever song streaming capabilities that has allowed them to become world-wide successes in the industry. It is only natural that I would want to learn from the best.

Let’s start by examining how these music streaming platforms handle the same issues that I have encountered while working on HeardIT. Both Spotify and SoundCloud follow a similar main design principle when storing the song data:

* **File storage system** – for storing the actual song files
* **Metadata database** – for storing the information about the songs (metadata)

The **file storage system** stores and handles the actual song files. As we have established already, these files can be of various sizes and it can become challenging to store them. A file storage system, is a kind of database that is specifically focused on keeping only the song files. These kinds of databases ensure that music files are available for streaming and can handle the large scale of data involved, with redundancy and replication to prevent data loss. They can also have sophisticated integrated methods for distributing and extracting the files stored within them. They are also designed to be scalable and cloud native. Creating such a system from scratch is not worth in this case since there are already cloud file storage systems that can be utilized such as Google Cloud Storage or Amazon S3. Spotify and SoundCloud use services like Google Cloud Storage and Amazon S3 to keep their song files and provide their users with a pleasant experience.

The **metadata database** stores and handles all of the information about the songs, like the title, author, lyrics, album information and more. These databases do not deal with storing the actual files, thus allowing them to handle all of the information about the songs that needs to be stored. For these kinds of databases, Spotify and SoundCloud use relational databases like PostrgreSQL and NoSQL databases like Cassandra.

A good question that can be asked at this point is: “*Is it not possible to store all of the data in one database, instead of creating two different kinds of databases that have to be constantly kept up to date?”*

This is great question that I asked myself while doing the research on the topic. The conclusion that I came up to while reading on the subject is that storing data in a single database is impractical due to the different requirements for audio files and their metadata. Audio files are large binary objects that are typically accessed in a read-heavy pattern, where the same file might be streamed by many users concurrently. This is best handled by scalable cloud storage solutions like Google Cloud Storage or Amazon S3 since they are focus and optimized for large-scale, cost-effective storage with high availability. Metadata on the other hand, which is structured data like song and album titles, artist names and so on, requires complex querying and transactional capabilities supported by relational or NoSQL databases. These kinds of databases can support operations like searching and filtering, often involving joining multiple tables, performing aggregations, and supporting transactional consistency.

Separating these database systems allows for independent optimization and scaling, which in turn ensures better performance, cost efficiency, resource utilization, higher availability and better disaster recovery. For these reasons, some of the largest music streaming platforms like Spotify and SoundCloud separate their songs and metadata into different databases.

This has provided me with a good understanding of what I need to do in order to create a proper reliable, fast and efficient way to store song files and data. I split my databases into the two parts described above: **file storage management system** and **metadata database**.

To do this I decided that using Google Cloud Storage as my **file storage management system** would be a great opportunity to get to know how to use a real file storage system and gain experience working with the real product. It benefits me greatly since it allows me to store my song files securely and have fast and reliable access to them. Moreover, Google Cloud Storage has an integrated load balancer, that can automatically balance the load of file usage that makes my files storage management system scalable and even more reliable. Another factor for using Google Cloud is because it integrates well with my other services which are already deployed to Google Cloud.

For my **metadata database**, I decide that the best technology to use would be a MySQL database. I have plenty of experience creating MySQL databases that can perform fast while also being reliable. The database is containerized and deployed to my Kubernetes cluster as a part of my deployments. In the MySQL database I will keep all metadata about the songs which will allow me to have an efficient way to extract song information and also perform complex search algorithms.

With this I conclude the approach that I am going to use to store some of the most important information on HeardIT which is related to the songs.

Methods used:

* Literature study – this method was used to determine the how do file management systems and other databases work
* Problem analysis – this method was used to determine exactly the challenges with storing song files
* Available product analysis – this method was used to determine what platforms like Spotify and SoundCloud do to tackle this problem
* Document analysis – this method was used to find out how to implement Google Cloud Storage as my file management system

1. What is the most suitable approach for storing and handling user data?

The next aspect of data storing and handling that I need to address is the user information. This includes user credentials, used for logging in and accessing some of HeardIT’ s features, and account information that they can upload and edit. These can include personal details and other sensitive information that has to be stored safely and securely. Personal data that is not meant to be made public must be inaccessible by other users.

Since the application will have to handle these kinds of data, it implies that the storage and handling of the users’ information needs to be done with security as its top priority. From my experience with creating secure user authentication, I can say that it is a very complex process that can have many unknown vulnerabilities. I personally, have not had the opportunity yet to get in-depth knowledge about the ever-expanding field of cyber-security and possible dangers that I need to be aware of. Due to time constraints, it does not seem feasible to have to develop and test against every possible danger out there. As such I decided to hand my authentication, authorization and account database protection to a trusted source that specializes in the cyber-security field of user data handling.

**Auth0** is an identity management platform founded in 2013 by Eugenio Pace and Matias Woloski, later in 2021 acquired by Okta, providing authentication and authorization services. It provides user identity management with features like multifactor authentication, single sign-on and support for various identity providers (Google, Facebook, etc.). Auth0’s aim is to help developers integrate secure authentication while focusing on the core of their applications. For this reason, I decided to integrate Auth0’s capabilities in my application. This would allow me gain experience using an external authentication and authorization provider while also speeding up the process and allowing me to focus on the rest of my application.

Using Auth0 for user authentication and storage comes with a multitude of benefits. It ensures robust security with features like built-in multi-factor authentication and support for various identity providers like Google, Facebook and more. It also ensures maintained compliance with major regulations such as GDPR. Auth0’s infrastructure is scalable and designed for handling high traffic and automatically adjusting to meet the demand. Its platform enables easy and straightforward integration and supports modern authentication protocols like OAuth 2.0 and SAML which facilitates seamless integration with other services. It also provides customizable authentication flows and enables deep data analysis through built-in analytics. As a managed external service that focuses on the authentication and security of the users’ data, it reduces the operational burdens and complications by handling the maintenance, security updates and continuous improvements. With a very active and helpful community with plenty of documentation and support, Auth0 is a product that will greatly benefit my application by ensuring it has a safe, secure and well-maintained authentication, authorization and user storage database. It will also allow me to gain valuable experience using such a service which I see a beneficial part of my developer knowledge.

However, there are some drawbacks that need to be addressed. Since I am going to be using a third-party tool for such a vital of my application, it is expected that there can be some concerns. However, Auth0 has proven over the years that it provides adequate security measures and complies with the industry standards (GDPR). Its reputation as an industry leader and acquisition by Okta lead to its widespread adoption by organizations of all sizes and all around the globe. It ensures transparency, evidenced by regular audits, certifications and documentation of security practices. It also is committed to continuous improvements through regular updates and solutions. Because of these reasons, I believe I can trust Auth0 as my user authentication, authorization and data storage provider.

To summarize, for the data storage and handling of my user information, I am going to use Auth0 since it will allow me to have a modern and secure authentication and authorization service, while also providing secure database facilitation and management.

Methods used:

* Literature study – this method was used to determine how can I protect my users’ data
* Available product analysis – this method was used to determine what kind of authentication and authorization service providers there are
* Document analysis – this method was used for finding out how to integrate Auth0 into my application
* Ethical check – this method was used to determine the credibility of Auth0 and possible drawbacks of using a third-party service

1. What is the most suitable approach for storing and handling other kinds of needed data?

The final set of data that HeardIT is working with is the more general and less sensitive types of information. This includes published song metadata, playlist information, public comments and announcements and others. I have aggregated these kinds of data since they will follow a more general approach when it comes to storage, scalability and security. These types of data can be stored in relational databases like MySQL and can be safely imported into scalable deployments to meet the demand.

In fact, these types of databases are a great match for storing my metadata and public information since, a relational database like MySQL can be a great asset for performing complex search and recommendation algorithms. Relational databases are designed to be fast and efficient and can be

Using a MySQL relational database is advantageous due to its strong support for ACID compliance, ensuring data integrity and reliable transactions. It efficiently handles structured data with complex relationships and enables sophisticated querying through SQL, making it ideal for applications requiring robust data management and analysis. MySQL's scalability, performance optimization features, security capabilities, and extensive community support further enhance its appeal. Additionally, being open-source and cross-platform, MySQL offers cost-effectiveness and versatility, making it suitable for various applications, including web development, e-commerce, and enterprise solutions.

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. 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.

**References:**

* How does Spotify work? Spotify Tech Stack explored. (2023, October 11). Intuji. <https://intuji.com/how-does-spotify-work-tech-stack-explored/>
* Cloud storage. (n.d.). Google Cloud. <https://cloud.google.com/storage?hl=en>
* Auth0: Secure access for everyone. But not just anyone. (n.d.). Auth0. <https://auth0.com/>
* AuTH0: key features, technical overview, and alternatives. (2024, May 1). Frontegg. <https://frontegg.com/guides/auth0>
* OAuth 2.0 — OAuth. (n.d.). <https://oauth.net/2/>