UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS730 Assignment 2

Shazam Music Platform

Student number: u23912589

Clinton Mgoduswa

May 5, 2023

Contents

1 Introduction

Shazam is a music recognition application that uses audio fingerprinting technology to identify music tracks from audio samples. However There's system architecture that is designed to be scalable, flexible, and reliable, to handle millions of requests per day and provide accurate and timely results to users. This document outlines the architectural design specification for the Shazam system application. In midst, the document is intended to provide a detailed overview of the system architecture and design, focusing on the non-functional requirements, architectural patterns, styles, and constraints, actor-system interaction, deployment model, and technical requirements.

2 Non-functional requirements

Non-functional requirements refer to the characteristics of a system that define its overall quality attributes. In the case of the Shazam system application, the following non-functional requirements have been identified

2.1 Quality requirements

- **Performance:** This refers to how well the system performs in terms of music tracks recognition from audio samples,speed, response time, and throughput within a few seconds.
- **Reliability:** This refers to the system's ability to perform its intended function without failure or error, and to maintain a certain level of availability.
- **Availability:** The system should be available 24/7 without any downtime for maintenance or upgrades.
- **Scalability:** The system should be able to handle increasing traffic as the user base grows and to remain performant even as the system grows.
- **Security:** The system should be designed with security measures to prevent unauthorized access, use, disclosure, disruption, modification, data breaches, and malicious attacks.
- **Maintainability:** This refers to the system's ability to be easily modified, updated, or repaired without affecting the system's overall functionality.
- **Usability:** The system should be user-friendly and easy to navigate and also have the look and feel.
- **Flexibility:** This refers to the system's ability to adapt to changing requirements or circumstances, and to allow for future expansion or modifications.

2.2 Quantification of the identified quality requirements

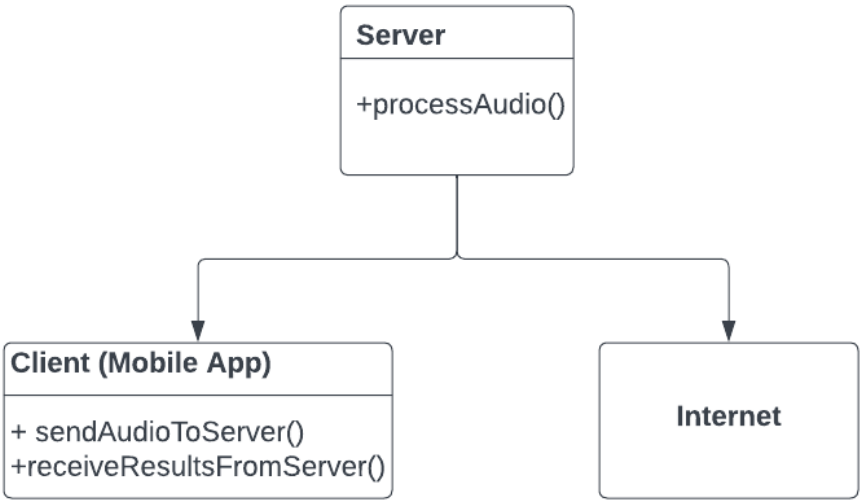
- **Performance:** The system should be able to recognize music tracks from audio samples within 3 seconds. In midst, the system should be able to handle a large number of requests concurrently with a response time of less than 2 seconds. The system should be able to handle 10,000 requests per second.
- **Availability:** The system should have a 99.9% uptime. The system should be able to recover from any failure or error within 5 minutes.
- **Scalability:** The system should be able to handle a growth of 50% in user base within the next 6 months. The system should be designed to scale horizontally by adding more servers, with a maximum of 5 servers to handle any growth.prevent unauthorized access.
- **Flexibility:** The system should be able to adapt to changing requirements without requiring significant changes to the codebase. The system should support configurable parameters to adjust system behavior, and the design should support modularity to enable easy replacement or addition of system components.
- **Security:** The system should use secure protocols to ensure that data is kept confidential and secure. The system should be able to detect and prevent any unauthorized access attempts. The system should adhere to industry-standard security best practices.
- **Maintainability:** The system should be designed with modularity in mind to make it easy to update and maintain. The system should have a maximum downtime of 1 hour per month for maintenance.
- **Usability:** The system should have a user satisfaction rating of at least 80%. In midst, the system should be tested for usability with both technical and non-technical users. The system should have a maximum learning curve of 30 minutes for new users.

3 Architectural design

The architectural design of the Shazam system application is based on the following components:

3.1 Architectural patterns

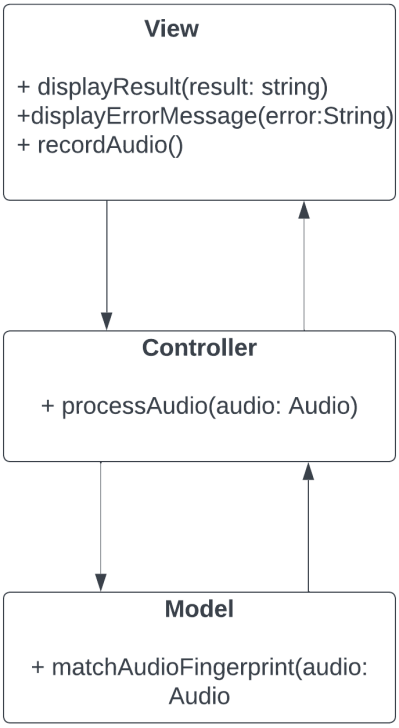
- **Client-Server Architecture:**



Client-Server Architecture diagram.

The diagram shows that the mobile app has two methods for communication with the server: sendAudioToServer() sends the audio fingerprint to the server for processing, and receiveResultFromServer() receives the result from the server. The server has a single method, processAudio(), which processes the audio fingerprint and returns the result to the client over the internet.

- **Model-View-Controller (MVC) Architecture:**

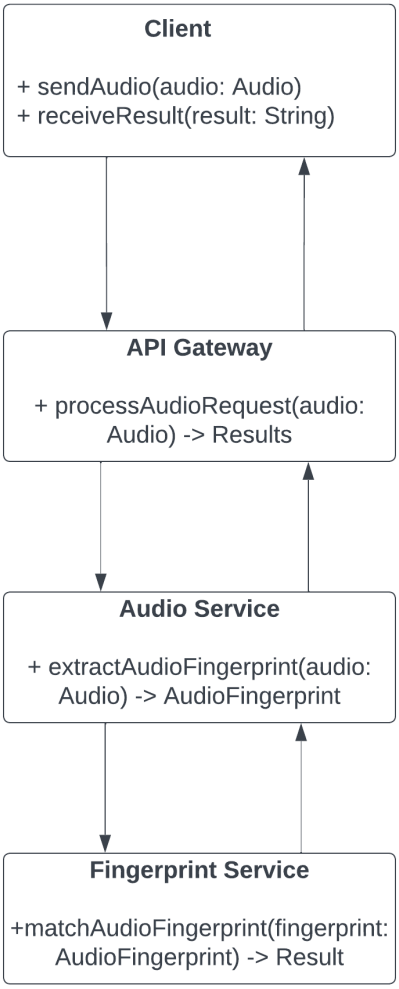


Model-View-Controller (MVC) Architecture diagram

This diagram illustrates that Shazam uses the Model-View-Controller (MVC) architecture to separate the user interface (View) from the business logic (Model) and the control flow (Controller). This separation allows for easier maintenance and development of the system.

It shows that the View communicates with the Controller by recording audio and displaying results and error messages. The Controller communicates with the View by sending the results and error messages to be displayed. The Controller communicates with the Model by calling the matchAudioFingerprint() method to match the audio fingerprint. The Model communicates with the Controller by returning the result of the matching process.

- **Microservices Architecture:**



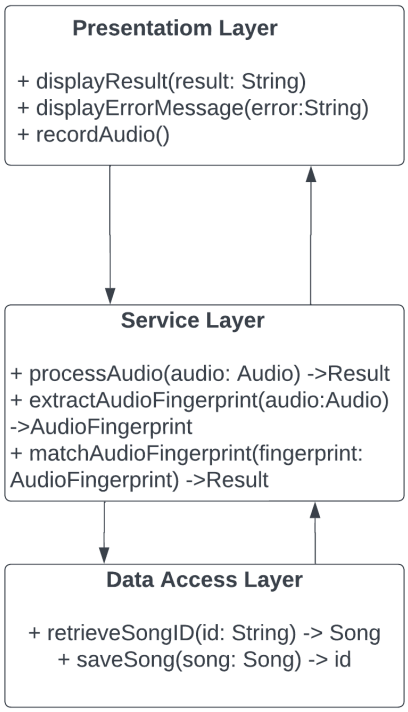
Microservices Architecture diagram

This diagram illustrates that Shazam uses a microservices architecture, where each component of the system is broken down into smaller independent services. The services are as follows:

- Client: This is the user-facing part of the application that sends audio to the system and receives results from the system.
- API Gateway: This component is responsible for receiving requests from the client and routing them to the appropriate microservice.
- Audio Service: This service is responsible for extracting the audio fingerprint from the audio file.
- Fingerprint Service: This service is responsible for matching the audio fingerprint to the correct song.

The diagram shows that the Client communicates with the API Gateway by sending audio and receiving results. The API Gateway communicates with the Audio Service by sending the audio to extract the audio fingerprint. The Audio Service communicates with the Fingerprint Service by sending the audio fingerprint to match it to the correct song. The Fingerprint Service returns the result to the API Gateway, which sends it back to the Client.

- **Layered Architecture:**



Layered Architecture diagram

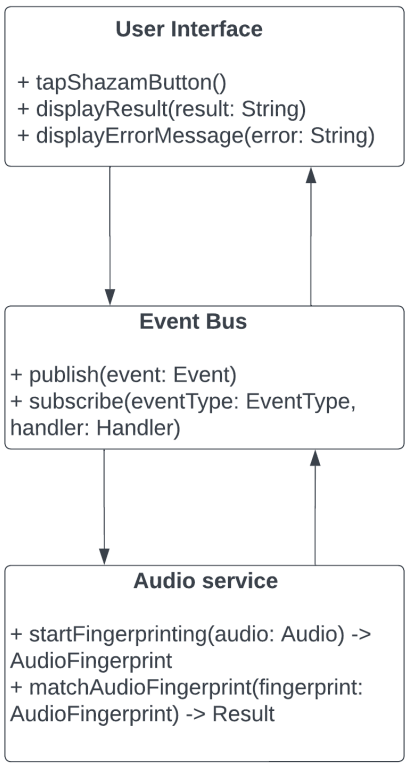
This diagram illustrates that Shazam uses a layered architecture, where different modules of the system are separated into distinct layers. The layers are as follows:

Presentation Layer: This layer is responsible for displaying results and error messages to the user, as well as recording audio.
Service Layer: This layer is responsible for processing the audio and extracting/matching the audio fingerprint.
Data Access Layer: This layer is responsible for retrieving and saving data (songs).

It also shows that the Presentation Layer communicates with the Service Layer by recording audio and displaying results and error messages. The Service Layer communicates with the Data Access Layer by retrieving and saving data. The Service Layer also communicates with the Presentation Layer by sending results and error messages to be displayed.

This layered architecture allows for easier modification of each layer without affecting other layers. For example, if the data storage system needs to be changed, the Data Access Layer can be modified without affecting the Service Layer or Presentation Layer.

- **Event-Driven Architecture:**



Event-Driven Architecture diagram

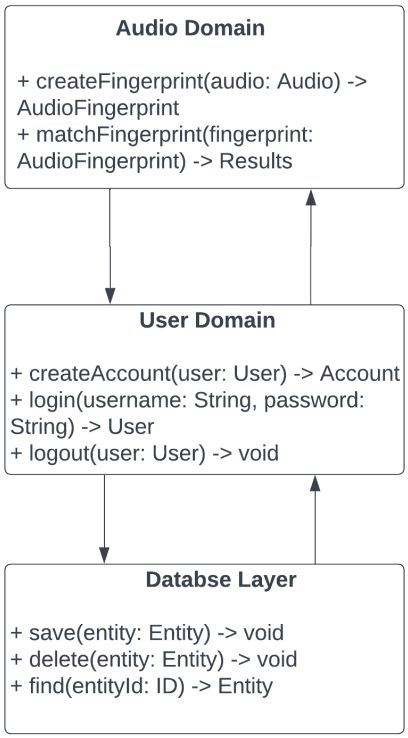
This diagram illustrates that Shazam uses an event-driven architecture, where events trigger certain actions within the system. The components are as follows:

User Interface: This is the user-facing part of the application that displays results and error messages to the user and allows them to tap the "Shazam" button to start the audio fingerprinting process.
Event Bus: This component is responsible for publishing events and subscribing to events. When an event is published, the appropriate handler is triggered.
Audio Service: This service is responsible for starting the audio fingerprinting process and matching the audio fingerprint to the correct song.

The diagram shows that when a user taps the "Shazam" button, an event is published to the Event Bus. The Audio Service has subscribed to this event and triggers the startFingerprinting method, which starts the audio fingerprinting process. Once the audio fingerprint is extracted, another event is published to the Event Bus with the fingerprint data. The Audio Service has also subscribed to this event and triggers the matchAudioFingerprint method, which matches the fingerprint to the correct song. Finally, the Audio Service publishes a result event to the Event Bus, which triggers the appropriate handler in the User Interface to display the result or error message.

This event-driven architecture allows for more loosely-coupled components and more flexibility in how the system can be extended and modified. It also allows for easier testing and debugging, as events can be simulated and tested independently.

- **Domain-Driven Design (DDD) Architecture:**



Domain-Driven Design (DDD) Architecture diagram

This diagram illustrates that Shazam uses the Domain-Driven Design (DDD) architecture, which models the business logic of the system around specific domains. The components are as follows:

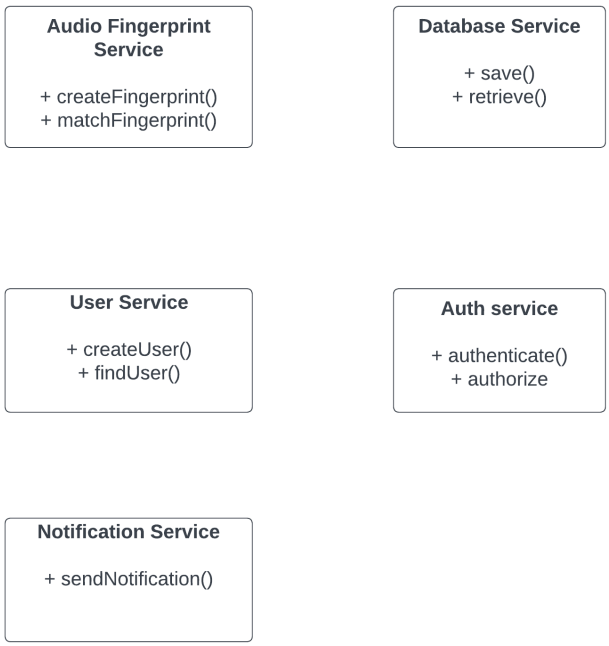
Audio Domain: This domain is responsible for creating audio fingerprints and matching them to the correct song.
User Domain: This domain is responsible for managing user accounts and authentication.
Database Layer: This layer is responsible for interacting with the database and performing CRUD operations on entities.

The diagram shows that the Audio Domain provides the createFingerprint and matchFingerprint methods, which are used to create audio fingerprints and match them to the correct song. The User Domain provides methods for creating user accounts, logging in, and logging out. The Database Layer is responsible for saving, deleting, and finding entities in the database.

This DDD architecture allows for easier development and maintenance of the system as the domain is the main focus. It also allows for easier testing

and debugging, as each domain can be tested independently. Additionally, it makes it easier to add new functionality to the system, as new domains can be added without affecting existing domains.

- **Service-Oriented:**



Service-Oriented diagram

This diagram illustrates that Shazam uses a service-oriented architecture where different services communicate with each other to achieve the desired functionality. The components are as follows:

Audio Fingerprint Service: This service is responsible for creating and matching audio fingerprints.
Database Service: This service is responsible for storing and retrieving data from the database.
User Service: This service is responsible for creating and finding users.
Auth Service: This service is responsible for authenticating and authorizing users.
Notification Service: This service is responsible for sending notifications.

The diagram shows that the Audio Fingerprint Service communicates with the Database Service to store and retrieve audio fingerprints. The User Service communicates with the Auth Service to authenticate and authorize

users. The Notification Service is responsible for sending notifications to users.

This service-oriented architecture allows for easier maintenance and scalability of the system. It also allows for better separation of concerns, as each service can be developed and maintained independently. Additionally, it makes it easier to add new functionality to the system, as new services can be added without affecting existing services.

3.2 Architectural style

3.2.1 Architectural Styles for Performance:

- **Concurrency:** Shazam could utilize concurrency to improve performance by allowing multiple audio fingerprinting requests to be processed simultaneously. This could be achieved through the use of multi-threading or parallel processing techniques.
- **Batch Processing:** Shazam could use batch processing to improve performance by processing audio fingerprinting requests in batches instead of individually. This would allow the system to process multiple requests at once, improving overall efficiency.
- **Scheduling:** Shazam could use scheduling to optimize performance by scheduling audio fingerprinting requests during off-peak hours or when system resources are not in high demand. This would help to reduce system overload and improve overall performance.
- **Caching:** Shazam could use caching to improve performance by storing frequently accessed data, such as audio fingerprints, in memory. This would allow the system to access the data more quickly, reducing processing time and improving overall performance.
- **Load balancing:** Shazam's legacy software may also use load balancing to distribute incoming requests across multiple servers, improving the system's ability to handle high traffic volumes and improve response times.

3.2.2 Architectural Styles for Security:

- **Authentication:** Shazam could use authentication to ensure that only authorized users are able to access the system. This could involve the use of username and password authentication, two-factor authentication, or other authentication methods.

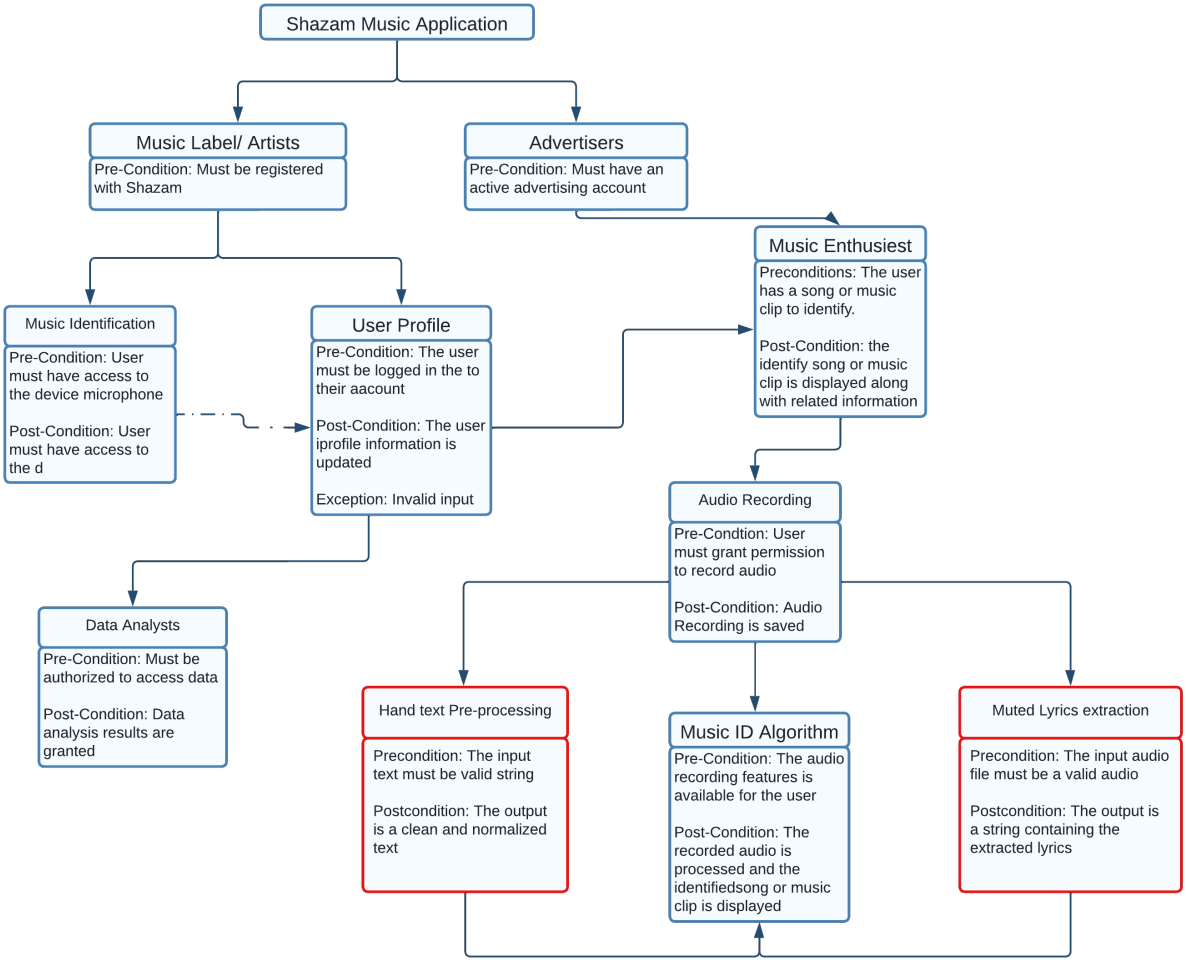
3.3 Architectural constraints

- **Hardware limitations:** The Shazam application needs to operate on a variety of mobile devices with different hardware capabilities. This requires

the design to be optimized for performance and resource utilization, taking into account constraints such as limited battery life, limited memory, and varying processing power.

- **Network latency:** Shazam relies on server-side processing to identify songs, which requires communicating with a remote server over a network. Network latency and bandwidth limitations may impact the system's performance and response times, requiring the design to incorporate techniques such as caching and load balancing to optimize performance.
- **Compatibility with different operating systems and devices:** The Shazam application needs to work on a range of different operating systems and devices, requiring the design to be platform-agnostic and adhere to industry standards and best practices to ensure compatibility and interoperability.
- **Privacy and security:** The Shazam application may collect personal data from users, such as their location and music preferences, which requires the design to incorporate robust privacy and security features to protect user data from unauthorized access or misuse. This may involve implementing encryption, access controls, and audit trails to ensure data privacy and security.
- **Regulatory compliance:** The Shazam application may be subject to various regulatory requirements, such as data protection laws and regulations, which require the design to incorporate features and functionality that enable compliance with these regulations.
- **Performance and scalability:** The Shazam application needs to be able to handle a large volume of requests and operate at scale, requiring the design to incorporate performance and scalability features such as load balancing, caching, and distributed processing.
- **Usability:** The Shazam application needs to be user-friendly and intuitive, requiring the design to incorporate features such as clear navigation, intuitive user interfaces, and responsive design to ensure a positive user experience.

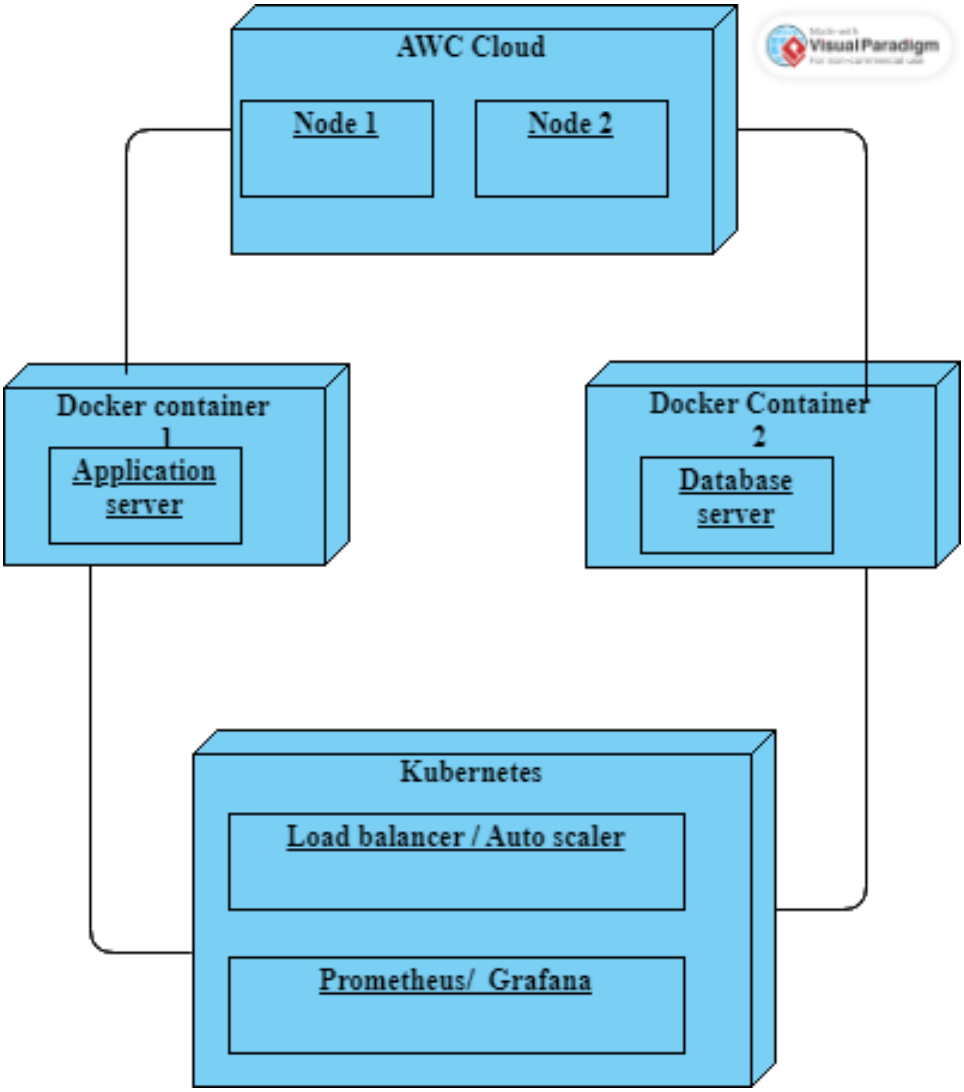
3.4 Actor-system interaction



Actor-system interaction diagram

3.5 Deployment model

• The system is deployed on cloud-based infrastructure using containerization technology.



Deployment Diagram

1. Use a cloud-based infrastructure such as AWS to host the system.
2. Utilize containerization technology such as Docker to package and deploy different components of the system.
3. Use Kubernetes as an orchestration tool to manage and scale containers across multiple nodes in the cloud environment.
4. Implement load balancing and auto-scaling mechanisms to ensure high availability and performance of the system.
5. Use monitoring tools such as Prometheus and Grafana to track system metrics and diagnose issues in real-time.

This design would allow for a scalable, flexible, and reliable deployment

model that can handle millions of requests per day while providing accurate and timely results to users.

3.6 Technical requirements (technology)

Python is a versatile programming language with a wide range of libraries and frameworks that can be used for developing a variety of applications, including audio processing and machine learning applications.

There are several libraries and frameworks available in Python that can be used for audio processing, such as Librosa, Pydub, and Audiostream. These libraries provide functions for reading and processing audio files, extracting audio features, and performing signal processing operations.

For machine learning, Python has popular libraries such as Scikit-learn and TensorFlow, which can be used for training and deploying machine learning models. Shazam uses machine learning algorithms for audio fingerprinting and recognition, and Python can be a suitable language for implementing these algorithms.

4 Conclusion

In conclusion the system is designed with a microservices architecture, event-driven architecture, client-server architecture, RESTful API, mobile-first design, and cloud-based infrastructure. The deployment model uses containerization technology and is hosted on AWS using Docker and Kubernetes.

In summary, the Shazam Music Platform is a modern and scalable system that leverages cutting-edge technologies to provide users with a seamless and intuitive music recognition experience. The system's architecture is designed to be modular and flexible, allowing for easy maintenance and updates. The cloud-based deployment model ensures high availability and performance while minimizing operational costs. Overall, the Shazam Music Platform represents a significant advancement in music recognition technology and sets a new standard for user experience in this field.

5 Bibliography

-"Shazam: Music Discovery, Charts Song Lyrics." App Store, Apple, 12 Apr. 2023, https://apps.apple.com/us/app/shazam-music-discovery-charts-song-lyrics/id284993459. Accessed 4 May 2023.

-Fowler, Martin. "Microservices." martinfowler.com, 25 Mar. 2014, https://martinfowler.com/articles/microservices.html. Accessed 4 May 2023.

-Gamma, Erich, et al. "Design Patterns: Elements of Reusable Object-Oriented Software." Addison-Wesley Professional, 1994.

-"Introduction to Domain-Driven Design." Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice. Accessed 4 May 2023.

-"Kubernetes Documentation." Kubernetes, https://kubernetes.io/docs/home/. Accessed 4 May 2023.

-Newman, Sam. "Building Microservices: Designing Fine-Grained Systems." O'Reilly Media, Inc., 2015.

-"Prometheus Documentation." Prometheus, https://prometheus.io/docs/introduction/overview/. Accessed 4 May 2023.

-Shvets, Alexey. "A Comprehensive Guide to Docker Monitoring Containers Monitoring." Logz.io, 27 Sept. 2021, https://logz.io/blog/docker-monitoring/. Accessed 4 May 2023.