Technical Design Document

Name: Tony Jiang

Semester: 6

Class: RB04

Version	Date	Description
0.1	30 Sept 24	Initial documentation.
1.0	17 Oct 24	Update microservice architecture.
1.1	7 Dec 24	Change the version structure.
1.2	19 Jan 25	Add new version

Contents

Introduction

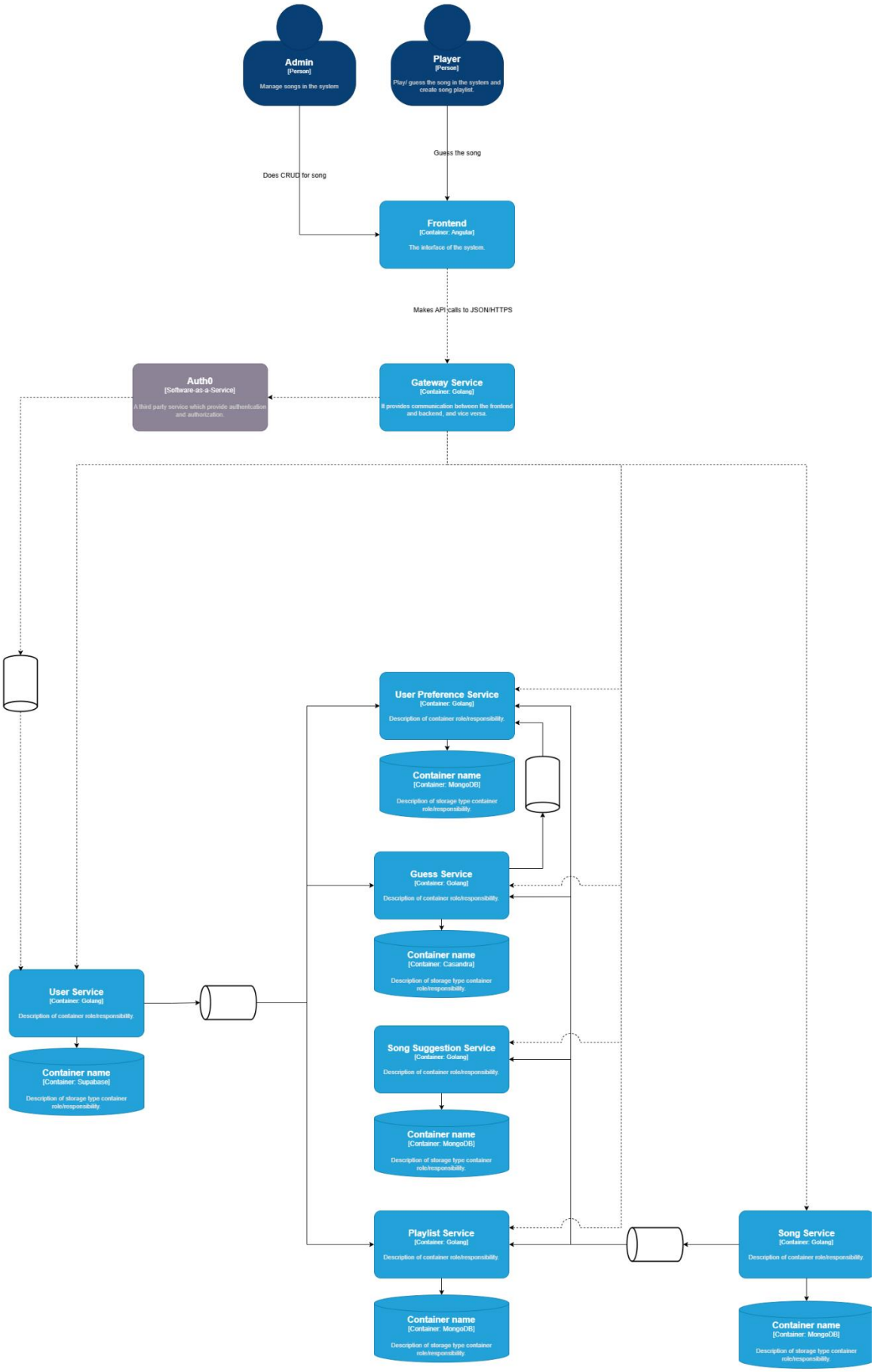
This section covers the technical aspects of the system, including its design and the reason behind the design decisions.

Microservice Architecture

The system is designed using a microservice architecture, allowing for scalable and flexible performance based on user demand. Each service can be independently scaled according to its usage, and services are decoupled, enabling changes to be made to one service without impacting others. This architecture also enables easier integration of CI/CD processes.

Version 3

This is the final version of the microservice architecture. I will only describe the new components here. If you want to know more about the other services, you can refer to version 2. Some services are still missing their database type because those services have not been implemented yet.


Admin
[Person]
Manage songs in the system
Player
[Person]
Play/ guess the song in the system and create song playlist.
Does CRUD for song
Guess the song
Frontend
[Container: Angular]
The interface of the system.
Makes API calls to JSON/HTTPS
Auth0
[Software-as-a-Service]
A third party service which provide authentication and authorization.
Gateway Service
[Container: Golang]
It provides communication between the frontend and backend, and vice versa.
User Preference Service
[Container: Golang]
Description of container role/responsibility.
Container name
[Container: MongoDB]
Description of storage type container role/responsibility.
Guess Service
[Container: Golang]
Description of container role/responsibility.
Container name
[Container: Casandra]
Description of storage type container role/responsibility.
User Service
[Container: Golang]
Description of container role/responsibility.
Container name
[Container: Supabase]
Description of storage type container role/responsibility.
Song Suggestion Service
[Container: Golang]
Description of container role/responsibility.
Container name
[Container: MongoDB]
Description of storage type container role/responsibility.
Playlist Service
[Container: Golang]
Description of container role/responsibility.
Container name
[Container: MongoDB]
Description of storage type container role/responsibility.
Song Service
[Container: Golang]
Description of container role/responsibility.
Container name
[Container: MongoDB]
Description of storage type container role/responsibility.

This version of the microservice architecture has changed slightly, with the addition of a new external service, the removal of the authentication service, and updates to some services with databases.

Auth0: This serves as the authentication service, eliminating the need to manually create and configure authentication and authorization. It also handles user creation, such as the sign-up functionality, instead of the user service. The user service focuses solely on managing user profiles. Auth0 is pre-configured; I only need to connect it to my other services. The Gateway API communicates with Auth0 through REST APIs. Auth0 communicates with the user service using a message queue to send user data to create the user in user service. Similarly, the user service uses a message queue to inform Auth0 when user data needs to be updated or deleted.

Databases

User Service: Uses a Supabase database, which is PostgreSQL-based and hosted in the cloud. This ensures consistency for user data.

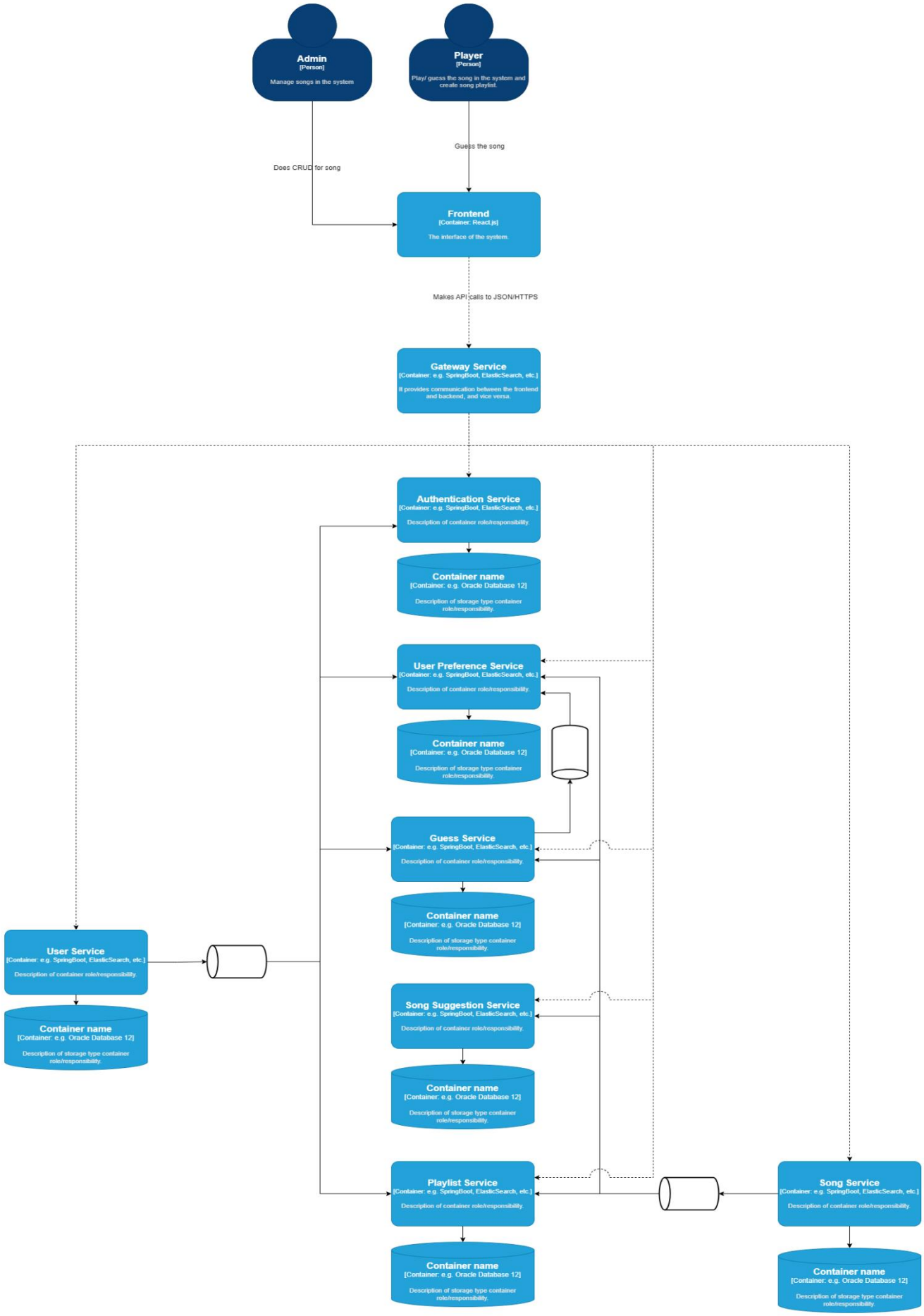
Auth0: Utilizes its own MySQL-based database.

Song Service: Uses MongoDB Atlas for scalability, as it is cloud-based.

Song Suggestion Service: Also uses MongoDB Atlas, primarily for scalability.

Version 2

This is the second version of the microservice architecture. The communication is clearly defined, but the databases for each service still need to be determined. Some aspects may change when programming begins.


Admin
[Person]
Manage songs in the system
Player
[Person]
Play/ guess the song in the system and create song playlist.
Does CRUD for song
Guess the song
Frontend
[Container: React.js]
The interface of the system.
Makes API calls to JSON/HTTPS
Gateway Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
It provides communication between the frontend and backend, and vice versa.
Authentication Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
User Preference Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
Guess Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
User Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
Song Suggestion Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
Playlist Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.
Song Service
[Container: e.g. SpringBoot, ElasticSearch, etc.]
Description of container role/responsibility.
Container name
[Container: e.g. Oracle Database 12]
Description of storage type container role/responsibility.

The microservice architecture is designed this way to be GDPR-compliant, allowing users the right to delete their information. When a user deletes their information, it should also be deleted across other services, such as the user ID.

Frontend: This is where the system's user interface is located, and where users interact with the system.

Microservices

Gateway services: It handles all API routes between the frontend and other services. It is primarily used to centralize and manage API routes, making it suitable for scaling systems when you have a lot of microservices in the system.

User service: This service manages user data and handles user sign-ups. It is connected to a database that stores all user-related information.

Authorization service: This service manages authentication and security, handling processes like user login. It is connected to a database that stores all authentication-related information about users. It also subscribes to a message broker from the user service. When a user is created, updated, or deleted, the user service sends an event to update this service with the relevant user information.

Song service: This service manages all the songs and is connected to a database that contains all the song information.

Playlist service: This service manages song playlists and tracks who created each playlist. It is connected to a database that stores all playlists. It subscribes to a message broker from both the user service and the song service. When a user is deleted, an event should be sent to remove the corresponding user ID from the playlists. Similarly, when a song is created, updated, or deleted, an event should be sent to update the relevant song information in the playlists.

Song Suggestion service: This service allows users to type in their answers, auto-filling suggestions based on their input to assist with their guesses. It is connected to a database that stores all auto-fill suggestions. It subscribes to a message broker from the song service. When a song is created, updated, or deleted, an event should be sent to update the relevant song information in the song suggestion service.

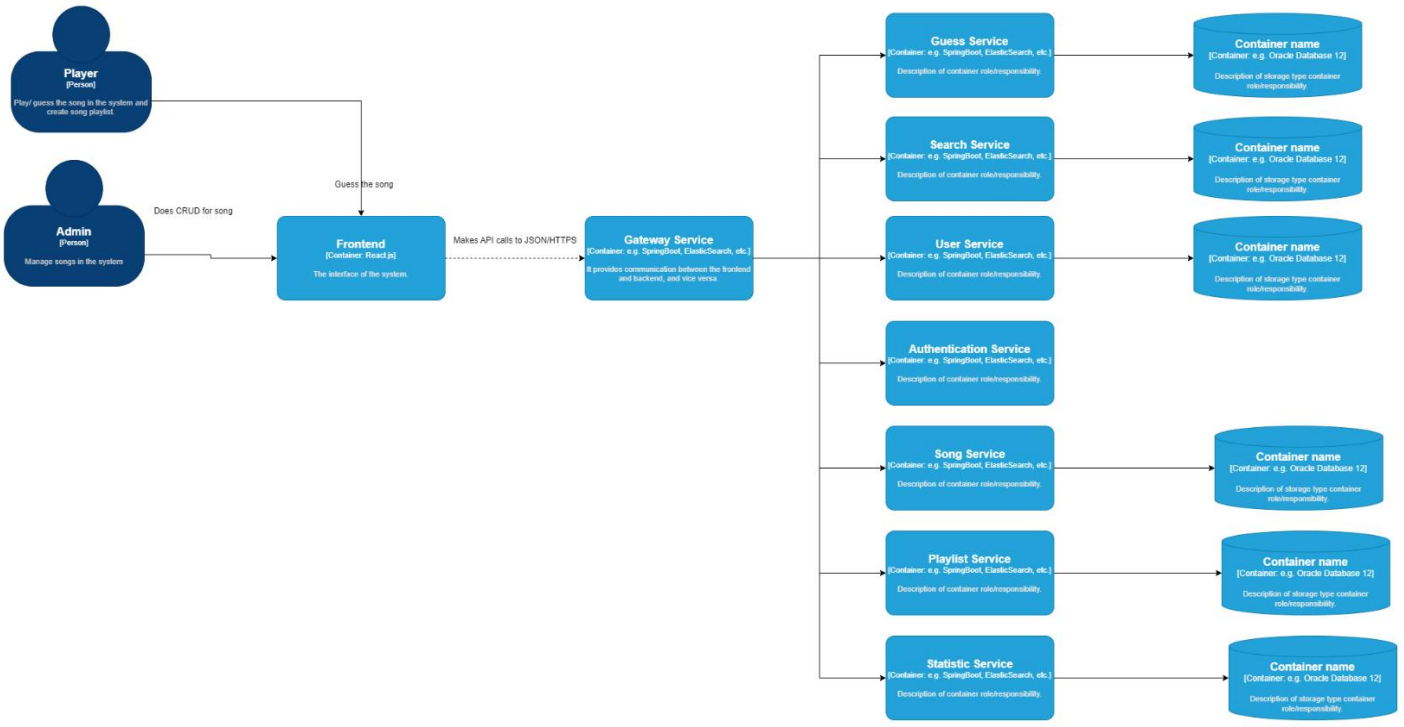
Guess service: This service compares the guesses made by users with the correct answer of the song. It is connected to a database to compare the correct answer with the user's answer. It subscribes to a message broker from both the user service and the song service. When a user is deleted, an event should be sent to remove the corresponding user ID from the system. Similarly, when a song is created, updated, or deleted, an event should be sent to update the relevant song information for the guess.

User Preferences service: This service tracks the songs that users like based on the correct answers given and the genres of the songs guessed correctly. It is connected to a database that stores user preferences. It subscribes to a message broker from the user service, song service, and guess service. When a user is deleted, an event should be sent to remove the corresponding user ID from the system. When a song is created, updated, or deleted, an event should be sent to update the relevant song information in the user preferences. When a user

guesses a song correctly or incorrectly, an event should be sent to indicate whether the user likes or dislikes the song.

Version 1

This is the first version of the microservice architecture. The microservice architecture is still in working progress, nothing is concrete yet and there is still missing some microservice communication with each other.



Frontend: This is where the system's user interface is located, and where users interact with the system.

Microservices

Gateway services: It handles all API routes between the frontend and other services. It is primarily used to centralize and manage API routes, making it suitable for scaling systems when you have a lot of microservices in the system.

Guess service: This is where the system compares the guesses made by the user with the correct answer of the song. It is connected to a database to compare the correct answer.

Search service: This is where the user types in their answer, and it auto-fills suggestions based on what the user types to help provide answers for their guesses. It’s connected to a database to get all the auto-fill suggestions.

User service: This service manages user data and handles sign-up processes. It is connected to a database that stores all user information.

Authorization service: This service manages authentication and security, handling processes like user login. It must connect to the user service to access user information.

Song service: This service manages all the songs and is connected to a database that contains all the song information.

Playlist service: This service manages song playlists and is connected to a database that contains all the playlists.

Statistic service: This service manages all the statistics and is connected to a database that contains all the statistical data.