# Technical Design

Name: Tony Jiang

Semester: 6

Project: Music trivia game

| Version | Date | Notes |
|---|---|---|
| **0.1** | 18 March 24 | Initial document. |
| **0.2** | 12 April 24 | Add C4 UML. |
| **0.3** | 19 April 24 | Add CI pipeline diagram. |
| **0.4** | 05 May 24 | Add yml file. |
| **0.5** | 24 May 24 | Update System Architecture |
| **0.6** | 11 Jun 24 | Update C2 and add Database section. |

# Contents

# Introduction

This is the technical design document for the music trivia web-based game. It includes all the technical details on how the project is implemented and its structure. This document also reflects on the design choices made within the project, helping to plan the configuration and address potential development challenges. Additionally, it aids in conveying the intended design to other developers, ensuring a shared understanding and agreement on the design approach.

# System Architecture

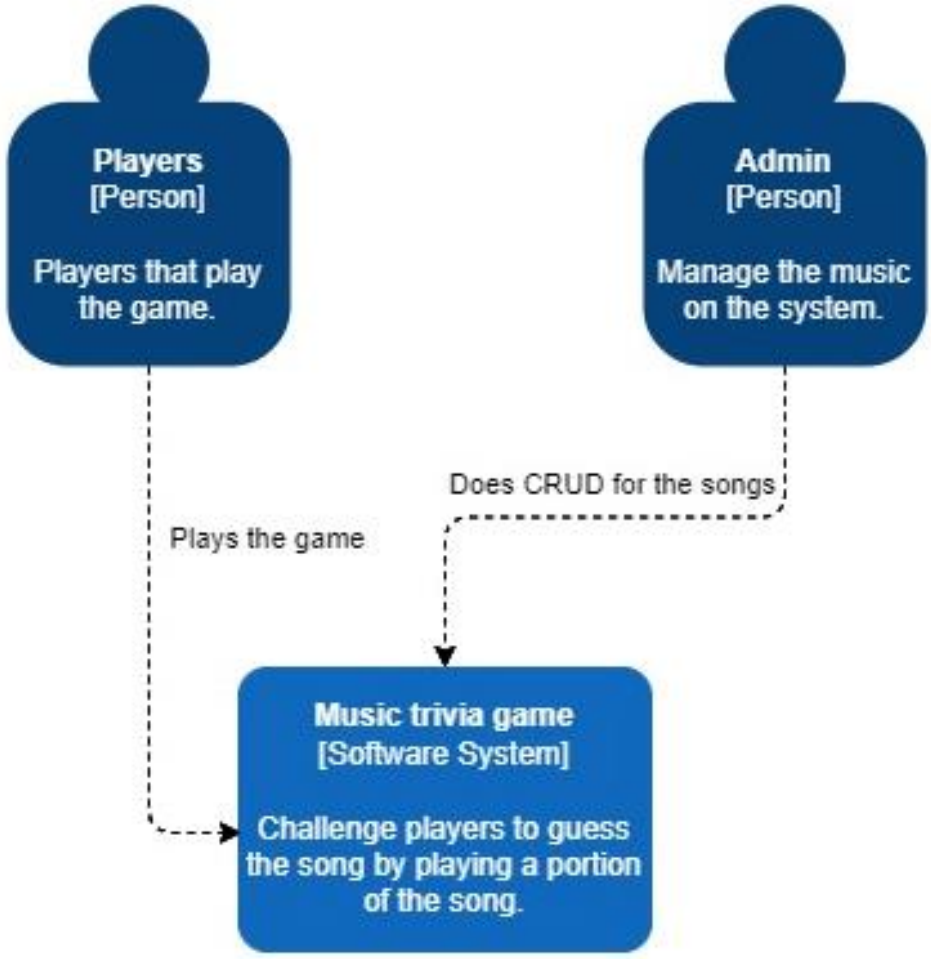
The purpose of System Architecture is to describe the internal system's overall structure and establish an agreement on the desired design of the system. To make it easy to describe and communicate the system's architecture, we'll use a C4 architecture diagram. It's an architecture design that is easy to understand. The design approach is straightforward and helps us communicate how each part of the system should be set up, even to a non-technical person.

The C4 architecture diagram has 4 level:

- Level 1: System Context (C1)
- Level 2: Containers (C2)
- Level 3: Component (C3)
- Level 4: Code (C4)

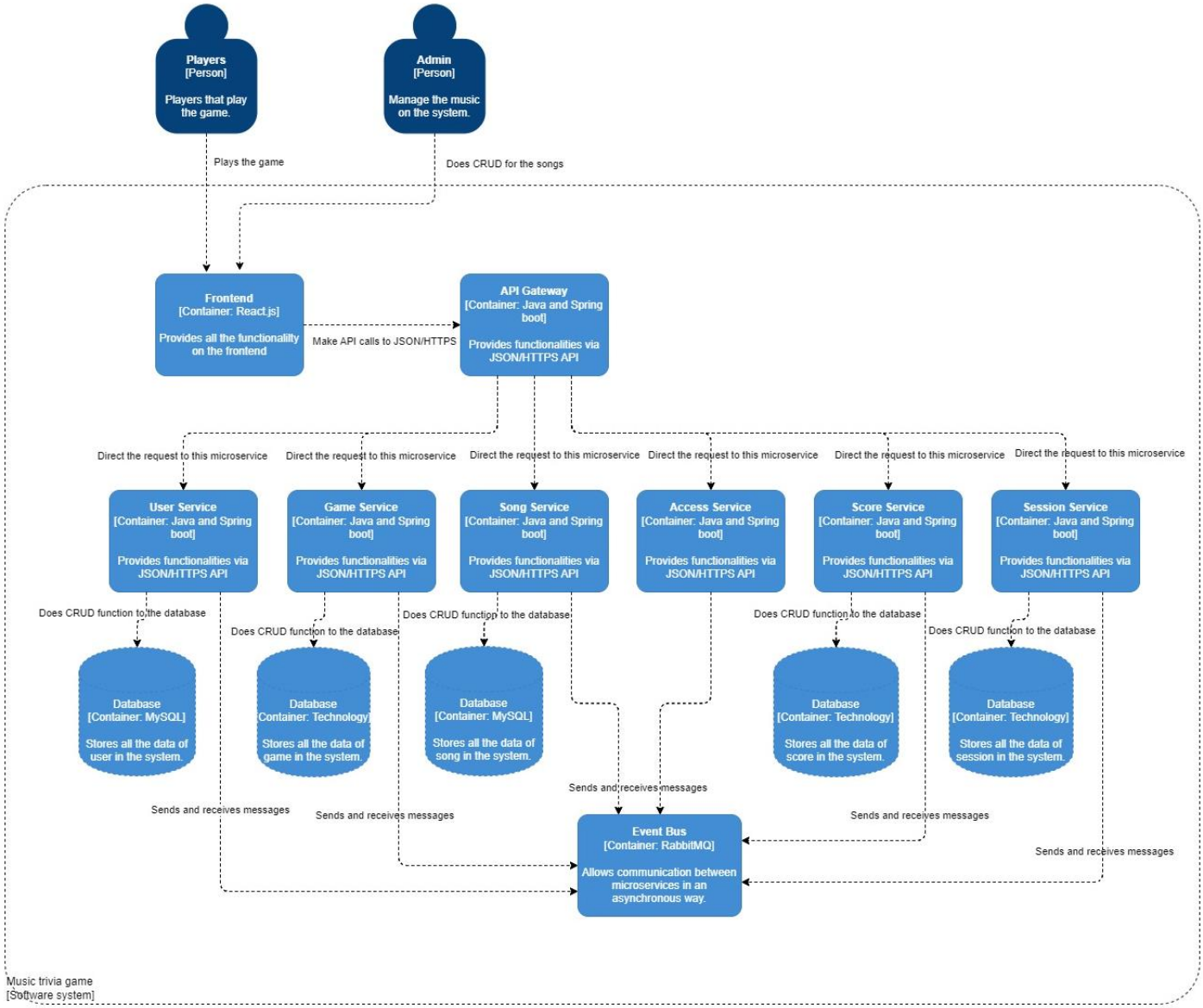
We will go to each level and describe what they do when we reach there. The tool that was used to create the C4 model is Visual Paradigm online. The free version.

## Level 1: System Context




In this C1 model, you can see that the player uses the music trivia system to play the game. The admin can also perform CRUD functions for the songs in the system. This model provides an understanding of which users can interact with the system, what type of system it is, and the actions they can perform.

## Level 2: Containers




As you can see from the diagram above, this represents the C2 model. The system consists of 14 containers, including:

**1 container for frontend.**

- **Description:** This container serves as the user interface, where users interact with the system. It communicates with the backend to process users' actions.

**1 container for API Gateway.**

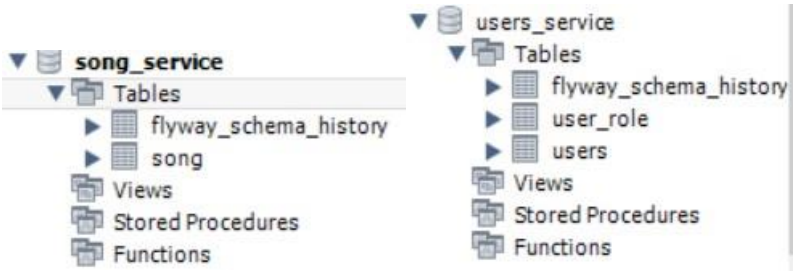
- **Description:** this container servers as communication from the frontend to the backend using API endpoint.

**6 containers for backend micro-service.**

- **Description:** These containers are the backend micro-service, each having the respective service/ functions of the system for maintaining and handle the project without any problems in individual setting and in group setting.

**5 containers for databases for each micro-service.**

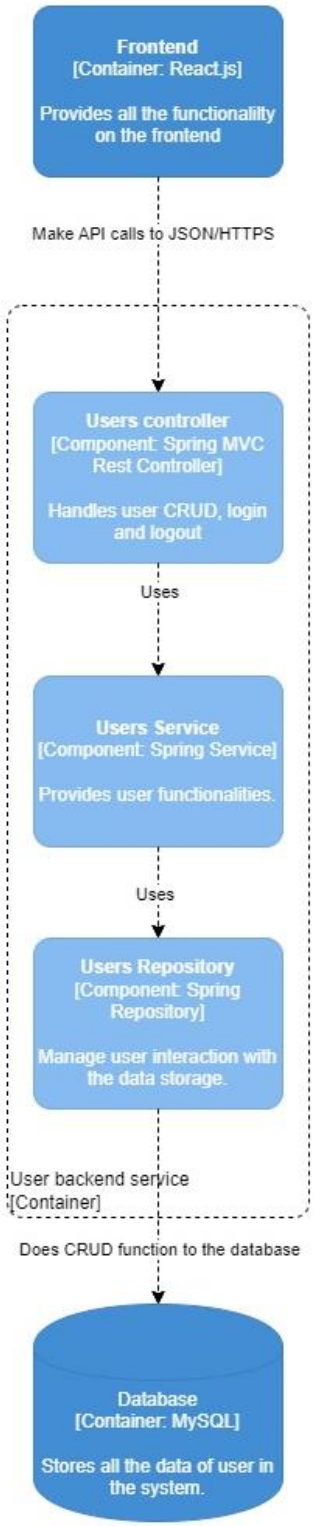
- **Description:** These containers serve as the databases for each specific microservice. Each database is assigned to a specific microservice for a particular reason. Some database technologies are not defined yet. Still in progress.
- These are the current micro-service that has a database.




**1 container for event bus**

- **Description:** This container facilitates communication between microservices by enabling them to send and receive messages.

## Level 3: Component




In the level 3 you have components. The components are the building blocks that make up the containers of level 2 and they interact with each other. These components are categorized by the function they are assigned to do.

This diagram illustrates the structure of the Backend container for the users service, which comprises three components, each performing distinct tasks. This division of tasks simplifies understanding of the design and facilitates the addition of future components. The naming convention “Users” is chosen to avoid conflict with a dependency that uses a function called “User”.

- Users Controller

This component serves as the endpoint for communicating with the Users Service.

It handles HTTP requests to the user function.

- Users Service

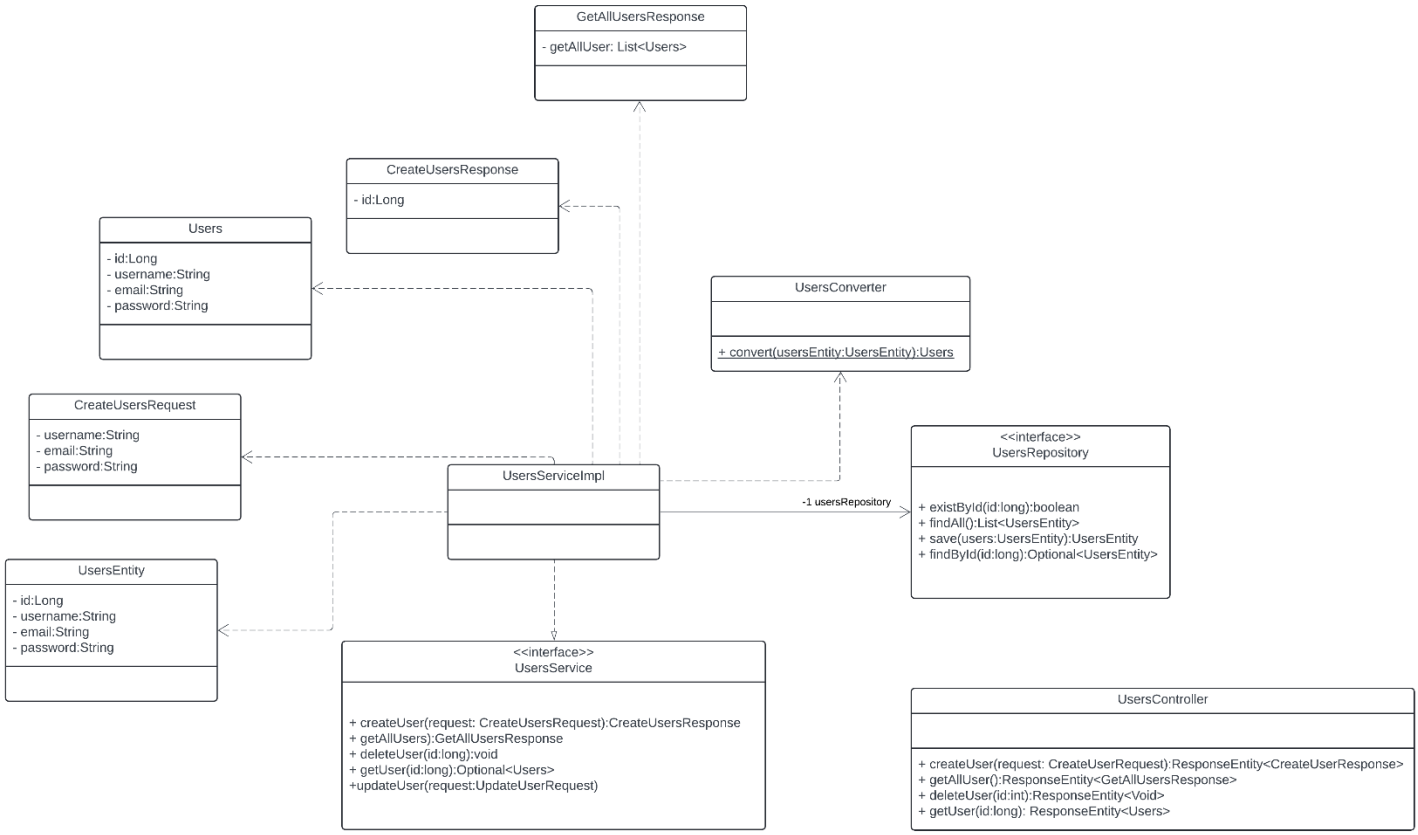
his component houses all user-related functions.

It interacts with the Users Persistence layer to retrieve user data.

- Users Persistence

Manages data flows from the database to specific user function.

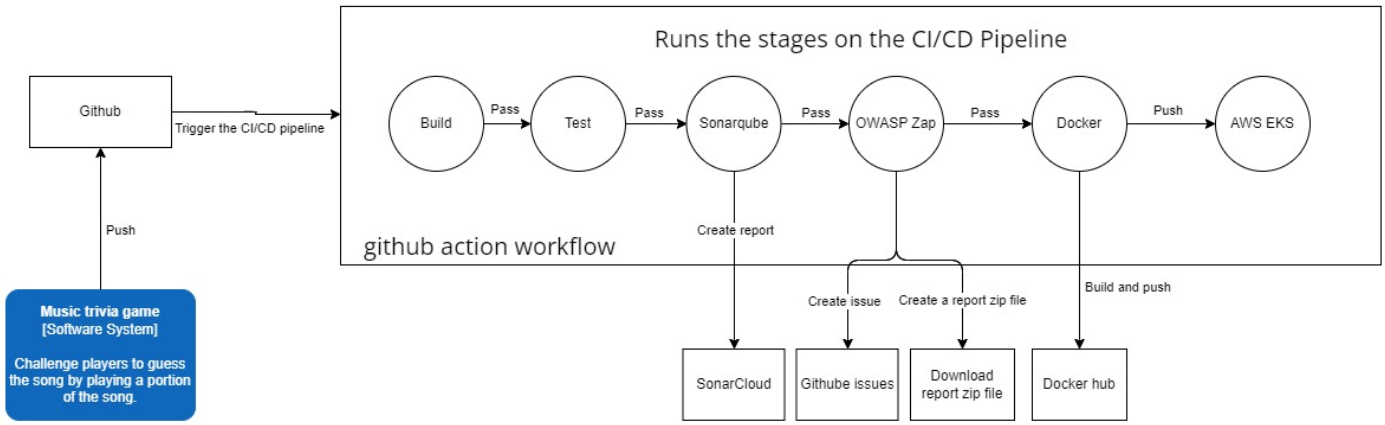
## Level 4: Code




At this level, only the complex UML diagram should be presented. This diagram is the structure of the User UML diagram, which serves as the foundational structure for every microservice. If any microservice has a different structure, it will be depicted in this level.

## CI/CD pipeline

This is the CI/CD pipeline sketch diagram on how it should look like and how it should work.




## YML file

This is the YML file on GitHub.

```yaml
name: MusicTrivia

on:
  push:
    branches:
      - main
      - development

  pull_request:
   types: [opened, synchronize, reopened]

permissions:
  issues: write

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-java@v4
      with:
        distribution: 'temurin'
        java-version: '21'
        cache: 'gradle'

    - name: Build with Gradle
      run: |
        cd MusicTrivia
        chmod +x gradlew
        ./gradlew assemble --no-daemon
```

```yaml

  test:
    runs-on: ubuntu-latest
    needs: build

    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-java@v4
      with:
        distribution: 'temurin'
        java-version: '21'
        cache: 'gradle'

    - name: Test with Gradle
      run: |
        cd MusicTrivia
        chmod +x gradlew
        ./gradlew test

  sonarqube:
    runs-on: ubuntu-latest
    needs: test

    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-java@v4
      with:
        distribution: 'temurin'
        java-version: '21'
        cache: 'gradle'
```

```
62
63        - name: Build with Gradle
64          run: |
65            cd MusicTrivia
66            chmod +x gradlew
67            ./gradlew assemble --no-daemon
68
69        - name: Sonarqube scan
70          run: |
71            cd MusicTrivia
72            chmod +x gradlew
73            ./gradlew sonar
74
75      owasp-zap:
76        runs-on: ubuntu-latest
77        needs: sonarqube
78
79        steps:
80        - uses: actions/checkout@v4
81
82        - uses: actions/setup-java@v4
83          with:
84            distribution: 'temurin'
85            java-version: '21'
86            cache: 'gradle'
87
88        - name: Build with Gradle
89          run: |
90            cd MusicTrivia
91            chmod +x gradlew
```

```
92            ./gradlew assemble --no-daemon
93
94        - name: Start the application
95          run: |
96            cd MusicTrivia
97            chmod +x gradlew
98            ./gradlew bootRun &
99          continue-on-error: true
100
101       - name: Run OWASP ZAP Baseline Scan
102         uses: zaproxy/action-baseline@v0.12.0
103         with:
104           token: ${{ secrets.GITHUB_TOKEN }}
105           docker_name: 'ghcr.io/zaproxy/zaproxy:stable'
106           target: 'http://localhost:8080/songs'
107           cmd_options: '-a'
108
109     docker:
110       runs-on: ubuntu-latest
111       needs: owasp-zap
112
113       steps:
114       - uses: actions/checkout@v4
115       - uses: actions/setup-java@v4
116         with:
117           distribution: 'temurin'
118           java-version: '21'
119           cache: 'gradle'
120
121       - name: Build with Gradle
```

```
122           run: |
123             cd MusicTrivia
124             chmod +x gradlew
125             ./gradlew assemble --no-daemon
126
127         - name: Set up Docker Buildx
128           uses: docker/setup-buildx-action@v3
129
130         - name: Login to Docker Hub
131           uses: docker/login-action@v3
132           with:
133             username: ${{ secrets.DOCKER_USERNAME }}
134             password: ${{ secrets.DOCKER_PASSWORD }}
135
136         - name: Build and push Docker image
137           run: |
138             cd MusicTrivia
139             docker build -t tonyj3/music-trivia-backend:latest .
140             docker push tonyj3/music-trivia-backend:latest
```

## Database

The database that is used for locally is MySQL workbench and to host the database on cloud I you AWS RDS. The main reason is because the cluster is built on and deployed on AWS and it's easy to connect it that way.