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

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

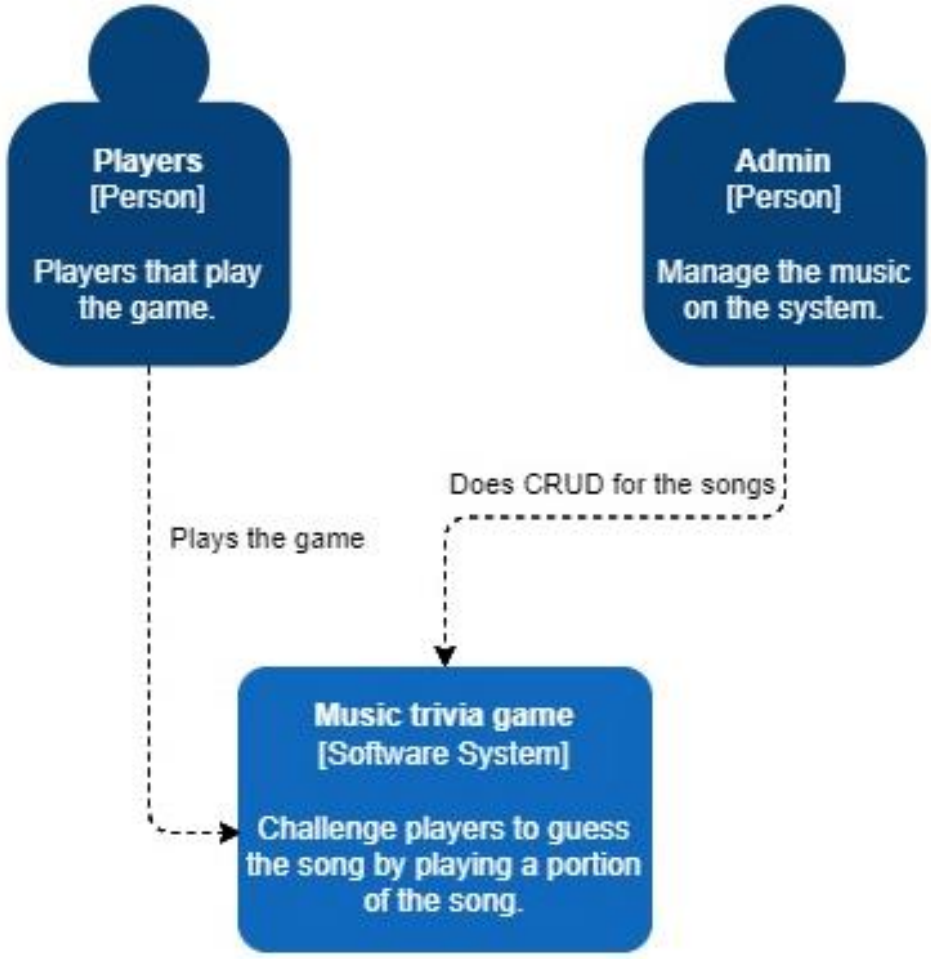
The purpose of System Architecture is to describe the internal system's overall structure and establish an agreement on the desired design of the system. To make it easy to describe and communicate the system's architecture, we'll use a C4 architecture diagram. It's an architecture design that is easy 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)

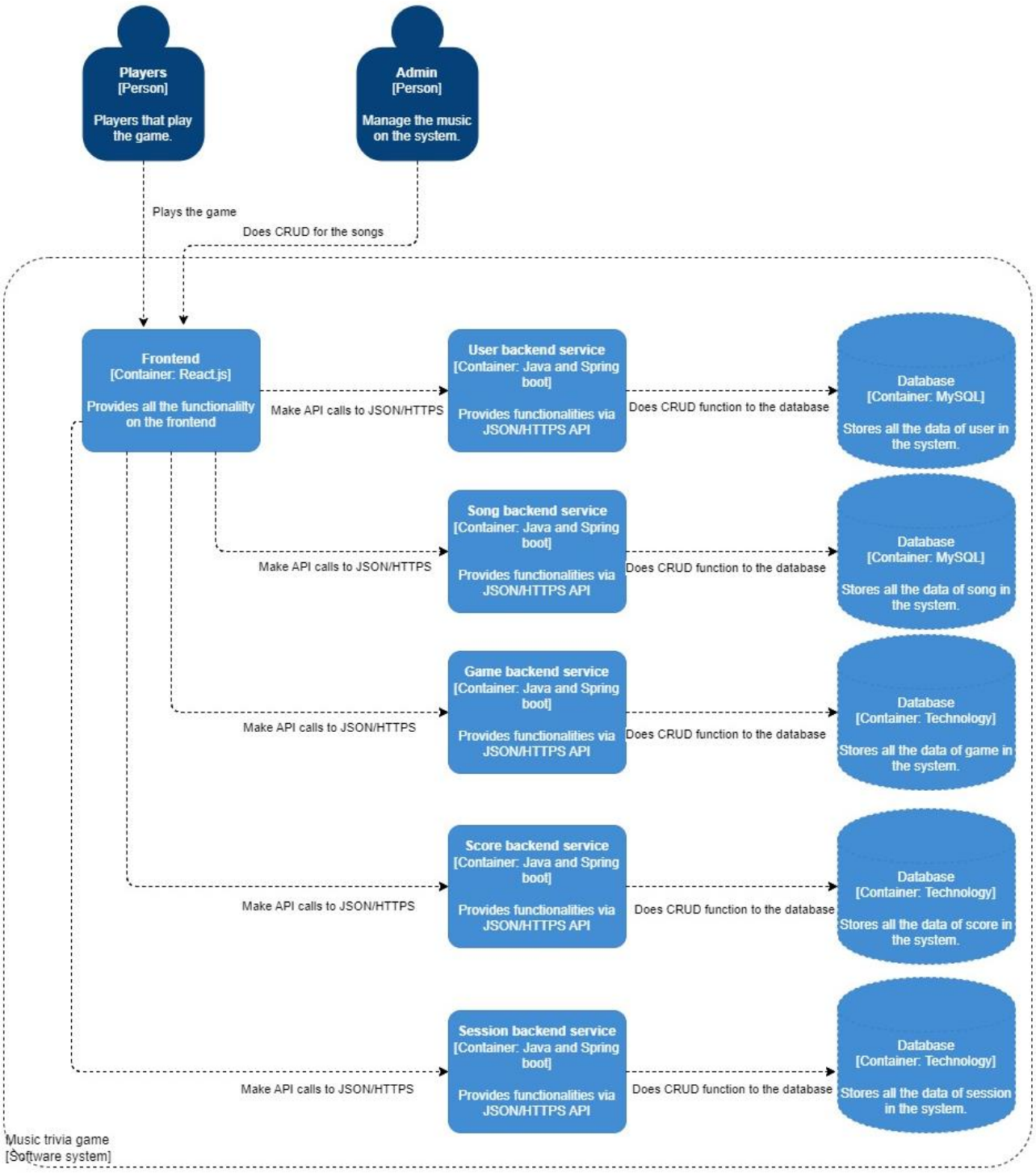
We will go to each level and describe what they do when we reach there. The tool that was used to create the C4 model is Visual Paradigm online. The free version.

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. However, the diagram is not yet complete; it is missing the communication between each micro-service. The system consists of 10 containers, including:

1 container for frontend.

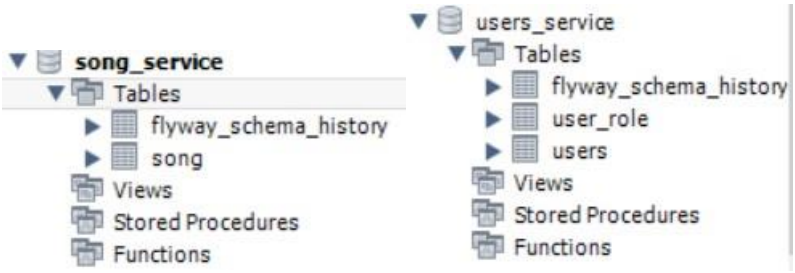
- **Description:** This component serves as the user interface, where users interact with the system. It communicates with the backend to process users' actions.

5 containers for backend micro-service.

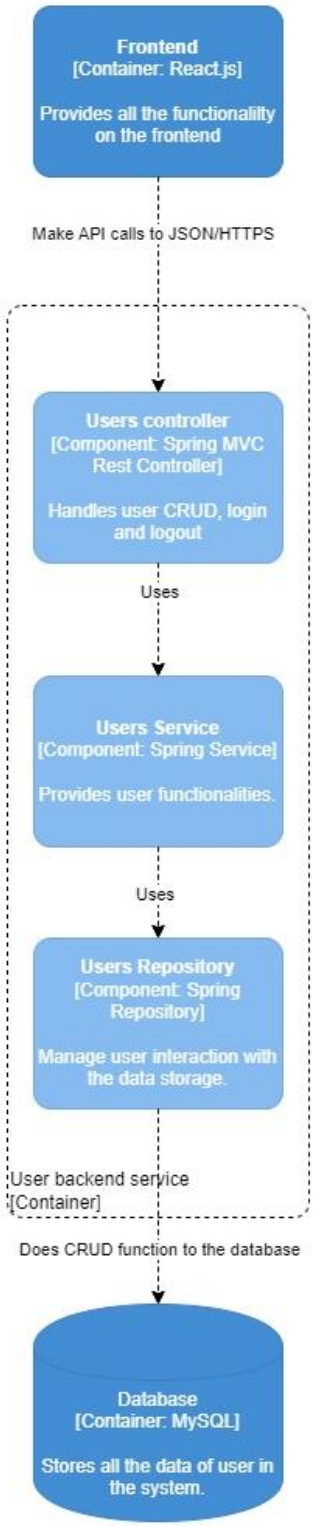
- **Description:** These containers are the backend micro-service, each having the respective service/ functions of the system for maintaining and handle the project without any problems in individual setting and in group setting.

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.



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

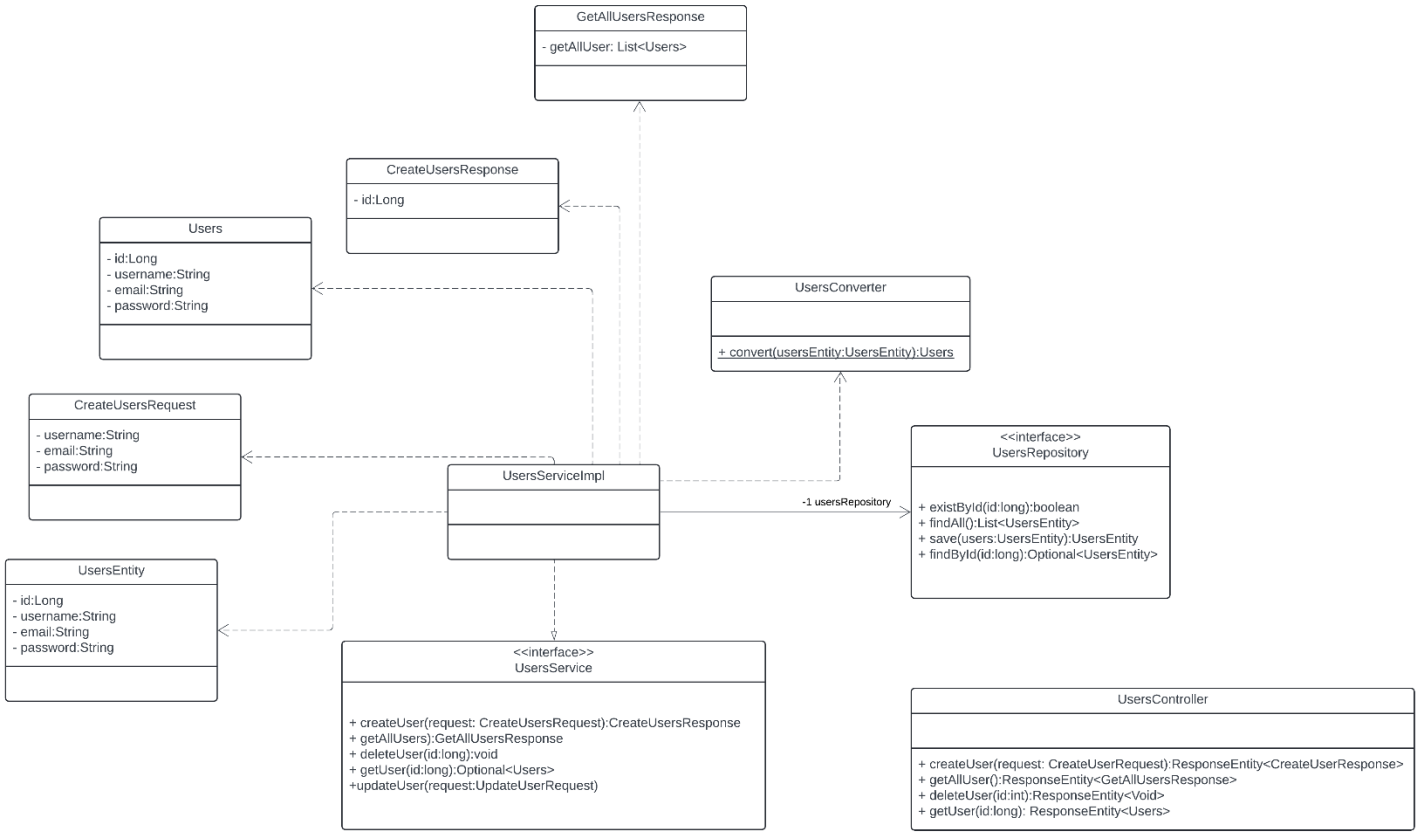
his component houses all user-related functions.

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

- Users Persistence

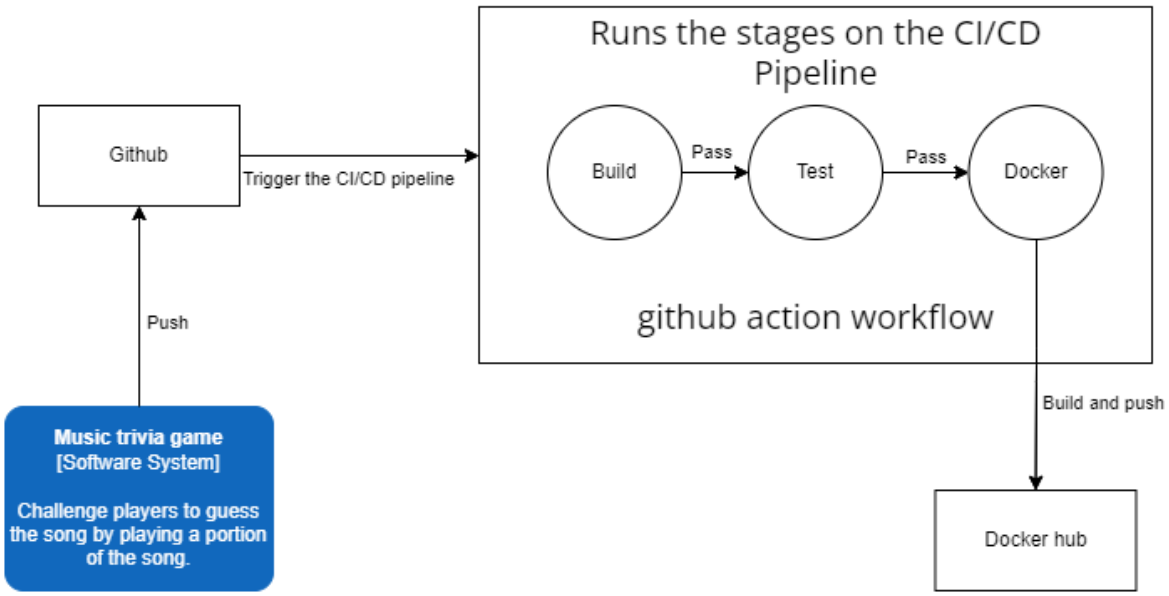
Manages data flows from the database to specific user function.

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



YML file

This is the YML file on GitHub.

Code | Blame | 76 lines (62 loc) · 1.54 KB

```yaml
name: MusicTrivia

on:
  push:
    branches:
      - development

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

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

    steps:
    - uses: actions/checkout@v4
```

```yaml
      - 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

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

    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
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and push Docker image
        run: |
          cd MusicTrivia
          docker build -t tonyj3/music-trivia-backend:latest .
          docker push tonyj3/music-trivia-backend:latest

```