THE GAMING ROOM

<Name-of-Software-Application>

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0 1.1 1.2	07/25/25	Trey Gault	Filled in required bracketed information.
	08/08/25	Trey Gault	Editing evaluation table.
	8/10/25	Trey Gault	Re edited evaluation table.
	8/19/25	Trey Gault	Edited recommendations

Instructions
Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

Executive Summary

We at CTS have taken on a new client, The Gaming Room, which wants to expand its Android only game. They want to expand their game Draw it or lose it a web-based/ multi-platform game. This will allow the game to reach a broader audience, The new application must ensure all teams, players, and games have unique identifiers to precent duplication and maintain constant data across platforms.

Requirements

Design Constraints

- Singleton limitation: only one instance of GameService should exist at a time
- Unique identification- every game, team and player need a unique ID, to prevent duplicates and maintain data integrity.

- Scale- Must support multiple concurrent games and teams on a network, allowing many users to use without issue.
- Maintainability- code should be modular and readable so that future updates and work can be done.

System Architecture View

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

Domain Model

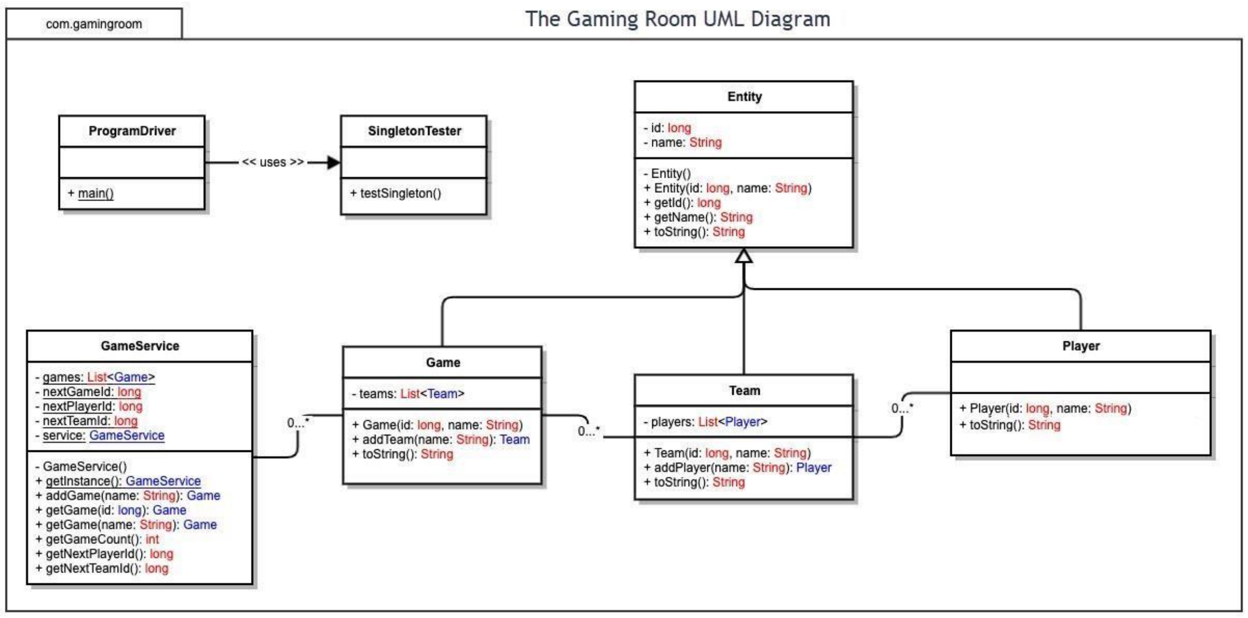
The UML diagram provided shows at its center is a “superclass”, “Entity”. Other classes like the Game, Team, Player classes, all inherit from the Entity class. This helps keep the code clean and repeat the same variables repeatedly.

Each Game class holds a list of Teams, and each Team Holds a list of players. This creates a clear path from the game to the teams playing and the players on those teams. This uses the common “has-a” principle. The game has teams, and the teams has players.

There is another class called Game Service. This is a Singlton, and it manages everything. A singleton means only one version of it exists in the memory while the program runs. This keeps everything in sync and avoids any mix-ups with duplicates.

Over this UML shows many key OOP principles.

- Inheritance- sharing code between classes
- Composition- Building bigger objects like Game out small ones like Team
- Encapsulation- keeping details inside classes



<u>Evaluation</u>

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client's requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

Development Requirements	**Mac**	**Linux**	**Windows**	**Mobile Devices**
Server Side	macOS can host our java app and works fine. But servers don't usually run apple hardware. No extra OS license, hardware wis pricey so its better for dev or testing than hosting.	Ideal for hosting.it runs java/dropwizzard app fine. Its free for OS license, and what most web servers use.	Windows works fine for hosting our java/dropwizzard. Windows has a built in web server to handle HTTPS and redirects, IIS Windows OS is paid. And servers require paid licensing.	Mobile devices are not designed for servers. Would not be a good fit.

Client Side Over all build one responsive web UI HTML/CSS/JavaScript) that talks to our java API follows web standards and test in major browsers.	Macs are great for running web apps, desktop apps and for development. Our UI works in safari/chrome. Since it's a webpage it doesn't require a macspecific build and will work with our app.	Users can access the web app and play it. To ensure compatibility with Linux we would use one web UI and make it responsive to fit different screen dimensions.	Users can access the Web app in any modern-day browser using windows. Most popular OS so lots of users.	It's extremely user friendly and the web app can be accessed through the phone's browser. This would require the most change to the UI. As it needs to support touch screen.
Development Tools	Java is used for building backend, dropwizzard, HTML/CSS/JavaScri pt for the UI, tools. Other tools needed like eclipse are free, so cost is low to none.	Java 11+, Maven, and dropwizzard run great on Linux. Free IDE like eclipse, IntelliJ, work just fine. And tools like Junit for testing. All this is free, so the cost is nea r 0.	Java, HTML/CSS and C# are all support Most tools work easily on windows. There are paid tool options like IntelliJ paid version or free ones like eclipse. Core dev tools are free but the OS cost.	Would use same code as desktop HTML/CSS/JavaScrip t. Use a Java Worker to cache files and let it run faster. Can still use eclipse, free, or VS code which is also free. We would have to use an emulator to test like android studios or Xcode.

<u>Recommendations</u>

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: I recommend Linux as the server platform for Draw It or Lose It. It will run java, drop wizard. It is already commonly used for web hosting and has no OS cost, so long term growth is cheap. Linux also already has systems in place for scalability.

2. **Operating Systems Architectures:** windows should run on a 64-bit architecture, which is standard for today. A 64- bit system allows for better performance and access to more memory, which is helpful for running larger games and handling many users.

3. **Storage Management**: We will need to store game data and images. For game data we would use a database like MySQL. This would include things like users, teams, games, rounds, guesses. Scores. Etc...

For images we would need to use object storage, we need to do this because of the size of the images, in terms of how much storage space they take up.

4. **Memory Management**: Linux Handles RAM fine, we will need to load only a select number of images at a time to maintain consistency in performance. Our app will keep the current and next image in memory and drop old ones.
5. **Distributed Systems and Networks**: We will run multiple copies of the game behind a load balancer(so servers don’t get overloaded). All players use the same HTTPS URL to get to game. Game state lives in the database not memory. If one server goes down others keep going. We can scale as player count grows.

   We will run our web-app as a web service behind HTTPS. Every device that connects to our game will access the same JSON. Servers won’t keep game data in Memory but instead to a database. We will run multiple game instances behind a load balancer, so no single server is overwhelmed.

6. **Security**:

Https- everywhere, encrypts traffic

Database encryption- to secure stored long-term data and be sure to secure backups.

Add accounts and logins, requirements for passwords hashed before storing

Add Token authentication.

Keep using our Roles properly like USER, and ADMIN roles.

Rate-Limit login attempts, to protect against brute force attempts.