

The Gaming Room

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324) **- 4**

[**Evaluation 4**](#_Toc115077325) **- 6**

[**Recommendations**](#_Toc115077326) **7**

## [Document Revision History](#_grjogdjh5fi8)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 11/17/24 | Adrian Tull | Filled in < > with proper information |
| 1.1 | 11/30/24 | Adrian Tull | - Corrected spelling and grammatical mistakes.  - Added context and clarifications to Evaluation |
| 2.0 | 12/15/2024 | Adrian Tull | - Corrected spelling and grammatical mistakes in the recommendation section.  - Added more context to recommendation and gave specific tool 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](#_sbfa50wo7nsh)

The Gaming Room wants to expand its Android based game, “Draw it or Lose It”, into a multi-platform, web based environment. This document outlines a solution to achieve this goal using object oriented programming and industry standard design principles. The solution uses a central GameService singleton to manage the games, teams, and players. The iterator pattern is used to ensure unique id naming and efficient managing of entities. This approach addresses the client’s requirements and prepare for further expansion and growth of the scope of the project.

## Requirements

1. The application must support multiple platforms beyond Android.
2. A game must consist of multiple teams, each containing multiple players.
3. Games, teams, and players must have unique identifiers and names.
4. Only one instance of GameService should exist in memory to manage all entities.
5. The application must efficiently handle duplicate checks for names and identifiers.

## [Design Constraints](#_2et92p0)

1. The application has to support multiple platforms while maintaining consistent data across sessions. The singleton pattern addresses these issues by ensuring a centralized management of data to avoid conflicts.
2. The game must handle increasing and decreasing numbers of users and teams without performance degradation. The iterator pattern is implemented to manage the entities increasing.
3. Unique ID and names prevent multiple people using the same name. There are checks to ensure that addGame and addTeam do not add a duplicate name.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

UML Description:

1. Entity base class encapsulates common attributes, id and name, shared by all 3 classes.
2. Game, Team, and Player all inherit from Entity, reducing the redundancy of code.
3. GameService is a singleton, ensuring that only one instance of GameService is running meaning there is a centralized management system for entitiy management.
4. The iterator pattern is used to traverse List<Game> and List<Team> for name and id uniqueness.

OOP:

1. Encapsulation was used to restrict direct access to attributes wile providing getter methods
2. Inheritance was used to promote code reuse and reduces duplication
3. Polymorphism allows the Entity class to have flexible implementation for derived classes.

"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.

## [Evaluation](#_2o15spng8stw)

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** | Macs are stable, but expensive. There is no support from Apple for a specific Mac Server OS anymore. Using a mac desktop OS could provide some challenges with scalability and system overhead | Linux is opensource, cost effective, scalable, and some distros are purpose built to be used as servers. Some distros have private companies backing them such as RedHat for support and frequent security updates | Windows server is great for enterprise hosting with a high level of support from Microsoft. The operating system alone is over $6,000 per license | Hosting on mobile devices in unfeasible. Mobile devices are not built to handle the amount of throughput and uptime. This option should only be considered with a serverless build or peer to peer connection. |
| **Client Side** | Developers can assume a relatively low variability of parts compared to the other options | A relatively low percentage of the population uses Linux for client side gameplay with the exception of SteamDeck which has a game compatbility layer that should help adapt web based development to SteamOS | Windows development can range from powerfule machines to lightweight, lower power devices. Extensive hardware support can become costly over time | Mobile devices are excellent for user accessability. Requires responsive design and cross platform compatibility. |
| **Development Tools** | Tools like Xcode if developing on a mac can provide Java and web development capabilities similar to most other IDEs | VS Code, and NeoVim both have support for web frameworks and server side development. For cross platform programs we can use Java to utalize the JVM on each platform. | VS Code, and NeoVim both have support for web frameworks and server side development. For cross platform programs we can use Java to utalize the JVM on each platform. | Androind Studio and Xcode are best suited for purpose built apps, but most other IDEs such as NeoVim using React Native and Flutter can reduce cost and allow the development team to do one mobile build rather than two. |

## Recommendations

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**: Linux based platform for server side hosting due to its stability, scalability and cost. It is lightweight, supports containerization with tools like Docker, and it can be spun up with cloud platforms like AWS for scalability. The open source nature of Linux allows for cost effective customization and frequent updates, making sure the platform remains optimized and secure.
2. **Operating Systems Architectures**: Microservices will allow the development team to have a modular development and allow scalability. Tools like Docker can be used to encapsulate each service, ensuring they run independently with minimal resource conflicts. Kubernetes can be used for orchestration, providing load balancing, automatic scaling, and fault tolerance, ensuring high availability even under heavy traffic.
3. **Storage Management**: Relational databases will ensure that we maintain data integrity and have fast queries for entity relationships. PostgreSQL has data integrity features and supports complex queries needed for managing entities like games, teams, and players.
4. **Memory Management**: Caching mechanisms will reduce the amount of server calls for frequently accessed game data and reduces database queries leading to improved gameplay performance. By implementing tools like Redis for inmemory caching, frequently accessed data (e.g., leaderboard or game states) can be retrieved faster, reducing latency.
5. **Distributed Systems and Networks**: REST APIs for communication between server and client devices will ensure platform independent communication, which is needed for this distributed system. For session consistency across distributed nodes, the application can use a distributed session store like Redis, making player experiences the same regardless of server location.
6. **Security**: HTTPS will ensure that any data shared between the game service and the client will be encrypted through a secure tunnel. The game does not currently gather user data, but if account creation is added later, there needs to be proper ways of handling that data such as encryption and safe storage of user data if that functionality is added later.