

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## 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)

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

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/26/25 | Valeria Duharte | Cover page name updated. Gave Executive summary, design constrain, and Domain Model. Evaluation page was updated as well as Recommendations. |
| 2.0 |  |  |  |
| 3.0 | 02/23/25 | Valeria Duharte | Updated recommendations information based on the operating system architectures for different OS, file components, memory management, systems and networks based on the client’s needs. |

**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 is the client. They are looking for a web-based game styled game show which they will call “Win, Lose or Draw”. The requirements of the client include must be multiple teams, each with multiple players, and with unique game and team names.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Security: The application will host random individual players of unknown size. Starting with security, allows for easier adaptation and streaming later.

Scalability: The game communicates over the web, so it has potential to grow into a large number of games, and users. The architect of the database should accommodate large numbers of current users.

Web-based environment: The game should be developed in a way that it runs good with common browsers and platforms.

## [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)

The class diagram shows the classes that exists in the game “Win, Lose or Draw”. The base of it all is the Entity class, since it contains entities like id and name, which are shared by the subclasses. The Game, Team and Player classes are subclasses that extend the Entity base class.

The GameService class has a composition relationship with the Game class, so it manages the lifecycle of the Game class, so it holds references to multiple Game objects. The Game class has a composition relationship with the Team class, and the Team class has a composition relationship with the Player class.

The ProgramDriver class is where the main entry point of the application is. Within the ProgramDriver class, the GameService singleton is created. The ProgramDriver class is responsible for adding games, teams, and players using GameService instance. The arrows show dependency of the Program Driver class on the SingletonTester class.

**"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** | Accessible and easy server. Flexible commands | Budget friendly but hard to navigate. | Expensive, user-friendly GUI. | Server specifications differ from user to user in mobile device. |
| **Client Side** | Expensive for users and need accurate skills to navigate the OS. Time and expertise required. | A lot more expertise and time required. To us the OS Linux data is needed. | Easy to use and operate. Minimum expertise required. | Flexible for clients to keep up with updates anywhere. More difficult to implement than other devices. |
| **Development Tools** | Consists of HTML, CSS, and JavaScript. Supports frontend developments. Includes PyCharm, GitHub, Visual Studios. | Consists of HTML, CSS, and JavaScript. Supports frontend and languages. Includes JavaScript, Ruby, PHP and Python. | Consists of HTML, CSS, and JavaScript. Supports frontend and languages. Includes Eclipse, PyCharm | Consists of HTML, CSS, and JavaScript. Programing languages consist of HTML, PHP, C++, and Python. |

## 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**: To use an operating platform, we need to take into consideration efficiency and stability, which it’s why Linux is the preferred choice. Linux is considered open-source and free to use, reducing costs. Unlike Windows, or macOS which are proprietary operating systems, meaning they are owned and controlled by a specific company. MacOS runs on Apple hardware.

Linux supports cross-platform tools and framework, so it’s easier to use across platforms like web, mobile, and desktop. This flexibility is a key factor in expanding to other computing environments. The Linux server is the most suitable because it supports scaling to other platforms and guarantee flexibility for expansions.

1. **Operating Systems Architectures**: Linux architecture features include.

* Monolithic kernel which includes all of the core system services like memory management, process scheduling, memory management, process management, and other functions in one block of code. In games like “Draw It or Lose It”, real time performance and responsiveness are crucial to ensure a successful game, and the monolithic kernel allows for high performance and efficient use of resources. One of the disadvantages is that if any system process or service fails, the entire system will crash.
* Process and Resources Management. Linux supports multithreading and multitasking and has an efficient process scheduler. Since the game is a multiplayer game, it will benefit having the ability to handle multiple current players interaction with minimal delay.
* Virtualization. Linux works with containerization technologies like Docker, which is used to build and deploy containers and facilitates the use of Kubernetes used for managing and scaling containers
* Security and Access Control. Linux’s security model includes user access control, process isolation, and robust firewall capabilities. It’s ideal for managing sensitive user data safely.

Linux’s monolithic kernel and its resource management capabilities will handle the game’s requirements and its ability to containerize facilitates smooth scaling and deployment.

1. **Storage Management**: The recommended storage System is MySQL Database with Amazon Relational Database Service (RDS)

Relational Database will be needed to manage user data and game state. MySQL is the best choice due to its robustness, performance, and support for structured data. MySQL provides

Data Integrity and Transactions by ensuring that the game remains consistent and reliable.

Cloud Database Management in a service like Amazon RDS supports high availability and replicas that will enhance performance and fault tolerance as the base grows. It helps by simplifying database management by automating tasks like backups, patching, and scaling. Amazon RDS allows for horizontal scaling, so it can be distributed across multiple instances to handle larger data volume. MySQL on Amazon RDS offers a secure, scalable, and reliable solution for managing the game’s data while supporting its performance and operational flexibility.

1. **Memory Management**: Linux Memory Management.

Linux uses virtual memory; it allows each space to have its own address space. Virtual memory ensure that multiple concurrent processes can operate without interfering with each other. It’s memory management automatically allocates and deallocates resources as needed. If the game experiences high demand and memory consumption goes up, Linux can swap out less active pages to ensure critical game processes remain active. It uses swapping to move data in and out of physical memories. It also uses caching mechanisms, that speed up the retrieval of frequently access data which improves the game’s performance. Linux’s memory management features virtual memory, swapping, and caching. It can handle high concurrency, larger user data, and smooth gameplay performance.

1. **Distributed Systems and Networks**: Distributed Architecture and Network Communication.

The game requires for real-time communication between players. WebSocket provides a communication channel that allows instantaneous data transfer between the client and the server. A microservices approach can be used to divide the game into smaller, independent services that interact with each other in the network. All of them could communicate via WebSocket connections. To handle potential connectivity issues, the architecture will be designed with redundancy and fault tolerance in mind. Load balancing, auto-scaling, and failover mechanisms provided by cloud platforms like AWS help us achieve this. Even in the case of minor network outages, data consistency is important to maintain the game synchronized across platforms. **WebSocket** for real-time communication, combined with a **microservices architecture** running on a cloud platform, will enable the game to function seamlessly across various platforms, ensuring low-latency and high availability. **Load balancing** and **auto-scaling** will provide fault tolerance.

1. **Security**: Securing user Data.

All communication between the client and server should be secured by SSL/TLS to protect user data and ensure secure gameplay.

SSL/TLS encrypts sensitive user information over the network like login, password, credentials, etc. We can also implement OAuth 2.0 or JWT (JSON Web Tokens) to secure authentication and authorization. This will allow users to securely log in and interact with the game without exposing their credentials directly to the server, ensuring that only authorized users can access their accounts and game data.

Sensitive data like passwords will be encrypted using modern cryptographic algorithms. Encrypted databases like Amazon RDS, will protect user information from unauthorized access. We will also implement role-based access control to restrict access to sensitive data and back-end management tools to ensure that only authorized users can perform certain administrative tasks.