

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

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

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/21/2025 | Cameron Sharp | Completed the executive summary, design constraints, requirements, domain model, evaluation, and recommendation. |
| 1.1 | 2/8/2025 | Cameron Sharp | Revised the table under evaluations to further align with what I know about Client, Server, and Development Tools. |
| 1.2 | 2/22/2025 | Cameron Sharp | Revised the recommendations section to align more with what I have learned about operating systems, architecture, storage and memory management, systems and networks, and security. |

**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](#_35nkun2)

The Gaming Room seeks to expand its popular Android game, Draw It or Lose It, into a web-based application accessible across multiple platforms. This transformation will broaden the game's reach, enhance user engagement, and provide a scalable platform for future growth. The web-based game will maintain the same core mechanics, allowing teams to compete in guessing puzzles rendered as stock images, with strict time limits for each round. To meet these goals, the proposed solution is designed around object-oriented principles, implementing a centralized GameService using the Singleton pattern to ensure a single active instance of the game. Unique identifiers for games, teams, and players ensure data integrity, while the system is structured for scalability and platform independence. This document outlines the software design, including its constraints, domain model, and implementation details, ensuring a robust and efficient solution for The Gaming Room’s needs.

## Requirements

The Gaming Room requires the application to support multiple platforms in a distributed web-based environment. Specifically:

* A game must allow for one or more teams to participate.
* Each team must have multiple players assigned to it.
* Games and team names must be unique, ensuring no duplication occurs when users create new entities.
* The system must ensure that only one instance of the game exists in memory at a time.
* Unique identifiers must be generated for games, teams, and players to avoid conflicts. These requirements are central to the design of the application and will be addressed using best practices in software engineering and object-oriented programming principles.

## [Design Constraints](#_1ksv4uv)

The development of the game in a distributed web-based environment introduces several design constraints. First, the application must support concurrent user interactions, ensuring seamless access and synchronization across multiple devices and platforms. To achieve this, mechanisms for thread safety and consistent state management are critical. Second, the Singleton pattern is required for the GameService class to guarantee that only one active instance of the game exists, ensuring centralized management and avoiding conflicts. Third, the system must enforce unique identifiers for games, teams, and players, which necessitates robust validation mechanisms to prevent duplication. Additionally, the application must be designed for platform independence, functioning seamlessly on Windows, macOS, Linux, and mobile devices. Finally, security is paramount; user information and game data must be protected from unauthorized access through encryption and secure authentication. These constraints influence the system's architecture, ensuring it is scalable, reliable, and capable of meeting The Gaming Room’s business objectives.

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

The Gaming Room application's well-organized object-oriented design is depicted in the UML class diagram. Game, Team, and Player all use the Entity class as a foundation, combining common properties (id and name) and methods like toString(). This encourages consistency and code reuse among all entities. The Singleton approach is used in the implementation of the GameService class to guarantee that there is only one instance, centralizing game management. In order to ensure data integrity, it keeps track of game lists, creates distinct IDs for each entity, and offers ways to add and retrieve games by name or ID.

In order to illustrate composition relationships, the Game class represents a single game and contains a list of Team objects, whereas Team has a list of Player objects. In order to prevent duplication, each class uses validation to enforce unique names for their respective entities. By using 0..\* multiplicities to indicate these relationships, games can have numerous teams and teams can have multiple players.

Encapsulation and other object-oriented concepts safeguard internal state, allowing functionality to be revealed only via regulated methods like addGame and addTeam. Polymorphism enables consistent behavior across all entities, while inheritance through the Entity class minimizes repetition. When paired with the Singleton pattern, these guidelines allow for scalable, stable, and effective system architecture that satisfies the client's needs for platform compatibility, unique identifiers, and single-instance management. The UML diagram ensures flexibility for future improvements while offering a solid implementation plan for the game application.

**"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](#_z337ya)

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 offers stability but limited adoption in server environments. It lacks widespread support for scalable, web-based hosting. Licensing costs are high due to expensive hardware. | Linux is the preferred server OS due to its stability, scalability, and open-source nature. It supports most web frameworks and has no licensing costs.. | Windows servers offer strong enterprise support, particularly for .NET applications, but have higher licensing costs compared to Linux. | Mobile devices are unsuitable for server-side hosting due to hardware limitations but require strong client-server communication for gameplay. |
| **Client Side** | macOS supports web browsers like Safari, Chrome, and Firefox. However, development costs are higher due to the need for specialized hardware and expertise in Swift.. | Linux supports modern browsers like Firefox and Chrome, but less user adoption makes testing essential. It offers low-cost development but limited user demand. | Windows is widely used, supporting all major browsers. Development is straightforward, but licensing and support costs can be higher. | Mobile devices are critical for gaming, requiring optimized web and native applications for both iOS and Android. Responsive design is essential. |
| **Development Tools** | Xcode and Swift are standard for macOS. IDEs like IntelliJ IDEA and frameworks like Spring Boot support web app development. High licensing costs for macOS hardware. | Linux supports open-source tools like Eclipse, IntelliJ IDEA, and Visual Studio Code. Node.js and Django are commonly used frameworks. No licensing costs. | Visual Studio is a powerful IDE for Windows development. .NET, Node.js, and Spring Boot are ideal for web-based applications. Licensing may apply for some tools. | Android Studio (for Android) and Xcode (for iOS) are essential. Cross-platform frameworks like Flutter and React Native reduce development time. |

## 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 is the most suitable operating platform for expanding Draw It or Lose It due to its cost-effectiveness, scalability, and compatibility with modern web technologies. Linux supports a variety of web frameworks (e.g., Node.js, Django, Spring Boot) and is compatible with cloud-based infrastructure like AWS, Google Cloud, and Azure. Additionally, there is no licensing cost making it the most cost effective option.
2. **Operating Systems Architectures**: The Linux-based server architecture follows a modular design, allowing efficient process management, multi-threading, and optimized resource allocation. It is lightweight compared to Windows, reducing system overhead and increasing performance. Most Linux distributions follow the monolithic kernel approach, improving I/O performance and security while ensuring better hardware interaction. Load balancing and redundancy mechanisms can be implemented easily to prevent downtime and ensure seamless gameplay.
3. **Storage Management**: Cloud-based storage solutions, such as Amazon S3, Google Cloud Storage, or Azure Blob Storage, are recommended because they offer scalability, redundancy, and high availability. Cloud storage lowers infrastructure maintenance costs while guaranteeing data integrity and automatic backups. The database should handle dynamic game interactions and player statistics using NoSQL (MongoDB, Firebase) and structured game-related data using MySQL or PostgreSQL.
4. **Memory Management**: Linux ensures effective memory consumption through the use of buffer caching, demand paging, and virtual memory management. In order to maximize resource efficiency, the Kernel Memory Manager dynamically allocates and deallocates memory for processes. Moreover, the caching and garbage collection systems in Node.js, Python, and Java.By releasing memory when it is no longer required, JavaScript enhances efficiency in Draw It or Lose It by avoiding memory leaks and needless lags.
5. **Distributed Systems and Networks**: The game should leverage a distributed architecture with WebSockets, RESTful APIs, and cloud-based networking solutions to allow for real-time cross-platform communication. To provide high availability, traffic will be divided across several servers using a load balancer. By lowering latency, CDNs (Content Delivery Networks) can guarantee quicker game updates. By guaranteeing that various game elements (leaderboard, game logic, and authentication) function independently, microservices-based deployment will enable modular scaling and lower the likelihood of system failures.
6. **Security**: To safeguard user data and stop illegal access, security is essential. Important security precautions consist of:

* Secure connection between the client and server is ensured via data encryption (TLS/SSL).
* User accounts are safeguarded and unwanted access is avoided with authentication and authorization (OAuth 2.0, JWT).
* Secures server access and stops harmful activity with firewalls and intrusion detection systems (iptables, Fail2Ban).
* Database security (encrypted storage, role-based access control): stops unwanted data changes and SQL injections.
* Frequent security audits and monitoring help to reduce possible risks by recording and tracking anomalous activities.