

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 11/17/2024 | Samim | Added Executive Summary, Requirements, Design Constraints, and Domain Model. |
| 1.1 | 11/30/2024 | Samim | Added the Evaluation section for development on different platforms. |
| 1.2 | 12/15/2024 | Samim | Added recommendations for an Operating Platform, Operating Systems Architectures, Storage Management, Memory Management, Distributed 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](#_sbfa50wo7nsh)

The Gaming Room, a company specializing in interactive gaming experiences, aims to expand its Draw It or Lose It game to a web-based platform, making it accessible across multiple devices. The current game is limited to an Android app, but the client recognizes the need for a broader, platform-agnostic solution to reach more users and enhance engagement. Addressing this platform shift is essential to achieving their business goals. The proposed solution will support unique game and team names, ensure only one instance of a game runs at a time using a singleton design pattern, and accommodate multiple teams with assigned players.

## Requirements

The Gaming Room’s web-based platform must support one or more teams in each game session, with each team accommodating multiple players. To ensure smooth operation, all game and team names must be unique, allowing users to check name availability when creating or joining a team. Also, the system must enforce that only one instance of the game exists in memory at any time, achieved through unique identifiers for each game instance, team, and player. The platform should be scalable and accessible across multiple devices and operating systems, ensuring a user-friendly and robust gaming experience. These requirements align with The Gaming Room’s goals of expanding their game to a broader audience effectively.

## [Design Constraints](#_2et92p0)

Developing The Gaming Room’s web-based game application in a distributed environment comes with specific design constraints derived from the requirements. First, ensuring unique game and team names requires implementing a centralized database or service to manage name validation in real-time, which could add complexity to the architecture. Scalability is another constraint, as the platform must handle varying numbers of teams and players without degrading performance, requiring load balancing and efficient resource allocation. Finally, the need for cross-platform compatibility introduces constraints on technology choices, ensuring the front-end and back-end solutions function seamlessly across different devices and operating systems. These constraints demand careful planning to balance performance, reliability, and scalability while meeting the client’s requirements.

## [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 UML class diagram represents a design for managing a game application, demonstrating the use of object-oriented programming (OOP) principles to meet software requirements. At its core, the GameService class employs a singleton pattern, ensuring only one instance of the game service exists in memory, as required. This class manages a collection of Game objects, each containing multiple Team objects, which in turn contain multiple Player objects. These hierarchical relationships are indicated by 0...\* associations. The Entity class serves as a base class for Game, Team, and Player, inheriting common attributes (id and name) and methods like toString(), promoting code reusability and consistency. Encapsulation is applied across all classes by making attributes private and providing public methods to access or modify them, safeguarding internal states and ensuring maintainability. Aggregation is demonstrated in the relationships between GameService and its dependent objects, allowing modular and independent management. Polymorphism is supported through method overriding, enabling flexible and context-specific behavior, such as customized toString() outputs for different 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** | Mac servers can host web-based applications, but macOS is not widely used for server environments. Hosting a web app on Mac requires a macOS server setup, which is less common and may lead to higher costs and maintenance. Licensing costs are generally higher compared to Linux, and scaling may be more difficult for a large user base like The Gaming Room's. | Linux is a popular choice for server hosting due to its stability, scalability, and open-source nature. It supports various web server technologies (Apache, Nginx) and is cost-effective since it has no licensing fees. Linux is highly scalable and well-suited for hosting large applications with a large number of players. | Windows Server can host web-based applications, but it comes with higher licensing costs compared to Linux. It supports IIS for web hosting and offers good integration with Microsoft technologies. Windows is a suitable option but may not be as cost-effective or flexible as Linux for large-scale deployment. | Hosting a web-based application on mobile devices is not practical due to hardware limitations and network dependencies. Mobile devices are best used for accessing the application rather than hosting it. The back-end server should be hosted on a traditional platform like Linux or Windows to handle traffic. |
| **Client Side** | For Mac, the app must be compatible with browsers like Safari and Chrome, requiring responsive design and cross-browser testing. Expertise in HTML5, CSS3, and JavaScript is needed. Development costs and time are moderate, with a focus on performance across different screen sizes. | On Linux, responsive design is also required for browsers like Chrome and Firefox. While Linux has a smaller user base, the open-source nature reduces restrictions. Expertise in HTML5, CSS3, and JavaScript is essential, with testing for consistency across distributions. | Windows clients need the app to work across browsers like Edge, Chrome, and Firefox. The app should be responsive, adapting to various screen sizes. Development is more time-consuming due to the wide range of devices, but expertise in web design and testing for browser compatibility is key. | For mobile platforms (iOS and Android), the app should be responsive and optimized for different screen sizes. Expertise in mobile-first design, touch events, and cross-browser compatibility is required. React Native or Flutter can be used to streamline development for both platforms. |
| **Development Tools** | Mac development requires tools like Xcode for iOS-specific features, but for web-based applications, IDEs such as Visual Studio Code and Sublime Text are commonly used. Languages like HTML5, CSS3, JavaScript, and frameworks like React or Angular are essential. The need for macOS-specific tools may increase costs and require a separate development team for iOS support. Licensing costs mainly involve Xcode for iOS development, but the rest of the tools are often free. | Linux supports open-source development tools like Visual Studio Code, Sublime Text, and Atom. Popular languages include HTML5, CSS3, JavaScript, Python, and frameworks like Node.js or Django. Linux is cost-effective since most development tools are open-source and free. The flexibility of Linux means that a single development team can often work across different platforms, reducing the need for multiple teams. | Windows supports IDEs like Visual Studio, which is commonly used for web development in C#, ASP.NET, and JavaScript. It also supports tools like Sublime Text and Visual Studio Code for front-end development. Licensing costs for Visual Studio can add up, but the tools are well-supported for large-scale development. Multiple teams may be required for managing different technologies and ensuring compatibility across platforms. | Mobile development tools vary between platforms. For iOS, Xcode is essential, while Android development often uses Android Studio or Flutter. For cross-platform apps, frameworks like React Native or Flutter can be used. These tools generally require separate development teams for Android and iOS unless using a cross-platform framework, which may impact cost and development time. Licensing costs are minimal for cross-platform tools, but IDEs like Xcode or Android Studio can involve platform-specific licensing. |

## 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**: I recommend using Linux as the server-side platform for expanding Draw It or Lose It to a web-based solution due to its scalability, cost-effectiveness, performance and exceptional stability. Linux’s open-source nature eliminates licensing fees, reducing costs considerably, while its stability and security make it ideal for handling high traffic loads. It supports essential web server technologies like Apache and Nginx and is compatible with cross-platform applications, ensuring accessibility for users on Windows, macOS, and mobile devices. Development is efficient with widely used, open-source tools such as Visual Studio Code, Python, and Node.js, allowing a single team to manage the project. Additionally, Linux excels in distributed environments, offering seamless integration with containerization tools like Docker and Kubernetes, which are fundamental for scaling the game’s infrastructure as its audience grows. This makes Linux the best choice for achieving The Gaming Room’s goals.
2. **Operating Systems Architectures**: The Linux operating system utilizes a monolithic kernel architecture, effectively managing critical functions such as process scheduling, memory allocation, networking, and file systems, ensuring both high performance and system stability. Its multi-user environment supports simultaneous user access, making it particularly suited for scalable applications like Draw It or Lose It. The modularity of Linux enables dynamic customization through kernel modules, allowing it to adapt seamlessly to specific operational requirements. Linux’s integrated support for networking protocols ensures efficient communication in distributed environments. Furthermore, Linux's compatibility with modern containerization technologies, such as Docker and Kubernetes, creates scalable deployment and resource optimization. This architecture makes Linux an ideal choice for The Gaming Room, aligning with the need for a secure, adaptable, and performant platform.
3. **Storage Management**: For the recommended Linux operating platform, an ideal storage management system would combine Logical Volume Manager (LVM) alongside a file system such as XFS. LVM offers dynamic storage management, enabling on-the-fly resizing of volumes and efficient disk utilization, which is important for supporting the scalability required by an expanding user base. XFS excels in handling large files and high-throughput environments, making it suitable for applications with significant scalability requirements. To further improve data security and reliability, implementing RAID (Redundant Array of Independent Disks) would add redundancy and fault tolerance, ensuring the protection of critical game data and user information.
4. **Memory Management**: The Linux operating platform utilizes advanced memory management techniques to ensure optimal performance and resource utilization, making it highly suitable for the Draw It or Lose It software. Through its virtual memory system, Linux combines physical memory (RAM) with disk-based swap space to provide applications with more memory than is physically available, allowing the game to handle concurrent users and large datasets efficiently. Techniques such as demand paging allocate memory only when required. Caching mechanisms store frequently accessed data in memory to reduce disk input/output operations and improve overall performance. Additionally, Linux employs kernel-level memory management to isolate processes, preventing memory leaks and ensuring system stability. These features collectively add to the platform's scalability, responsiveness, and reliability, enabling Draw It or Lose It to maintain operation even under heavy usage.
5. **Distributed Systems and Networks**: Draw It or Lose It employs a distributed system architecture to achieve seamless cross-platform functionality. This architecture employs interconnected network components within a client-server model, where a central server manages core game logic and data. At the same time, client applications on diverse user devices interact through RESTful APIs. This ensures platform independence and standardizes interactions between the server and clients. The system relies on network infrastructure, including high-speed internet connections and cloud services, to maintain reliable performance and minimize latency during gameplay. To address distributed system challenges, strategies such as load balancing and failover systems distribute traffic evenly across servers and redirect users to backup servers during outages. Secure communication is prioritized using encryption protocols like HTTPS, safeguarding data transmitted between clients and the server.
6. **Security**: Security is fundamental for ensuring user trust and data integrity and the Linux operating platform offers robust features to safeguard user information. Mandatory access controls (MAC), process isolation, and secure authentication mechanisms provide a strong foundation for platform security. Encryption protocols such as TLS/SSL ensure secure communication between clients and servers, protecting data in transit from unauthorized access or interception. On the server side, Linux supports strong user and role-based permissions, reducing the risk of unauthorized access to sensitive data and system resources. Additional measures like firewalls and intrusion detection systems can monitor network traffic and prevent cyberattacks. For stored user data, encrypted storage solutions such as LUKS ensure sensitive information remains secure, even if physical access to storage devices is compromised. Multi-factor authentication (MFA) further enhances account security across platforms. Together, these advanced capabilities make Linux an excellent choice for protecting user data on the platform and during communication across distributed systems.