

**Draw It or Lose It**

# **CS 230 Project Software Design**

Version 1.1

## 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 | 09/22/2024 | John Lopes | Initial draft of the software design document, including the Executive Summary, Design Constraints, and Domain Model sections. |
| 1.1 | 10/04/2024 | John Lopes | Evaluation of each Linux, Mac, Windows, and mobile platforms as operating systems. |
| 1.2 | 10/20/2024 | John Lopes | I provided detailed recommendations for the operating platform, system architecture, storage management, memory management, distributed systems, and security to support the expansion of "Draw It or Lose It" into a web-based, distributed environment. |

**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 aims to expand its mobile game, "Draw It or Lose It," into a web-based format to enhance accessibility and reach a wider audience. The primary software design challenge involves creating an application that allows multiple teams and players to engage in gameplay while ensuring the integrity of game states and user data. Our proposed solution includes implementing a singleton design pattern to manage game instances efficiently, allowing only one game instance in memory at any time. This design will facilitate seamless gameplay, minimize latency issues, and ensure a consistent user experience across platforms.

## Requirements

The client’s requirements for the game application encompass both business and technical aspects. From a business perspective, the primary goals are to expand accessibility of the game to web users, support multiple teams and players in real-time, and ensure unique identifiers for games, teams, and players. On the technical side, the application must be developed as a responsive web platform that is compatible with various browsers. Additionally, it is essential to implement secure user authentication and robust data management practices while optimizing performance to effectively handle simultaneous users. This comprehensive approach aims to create a seamless and engaging experience for all participants.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment presents several design constraints that must be addressed. First, network latency is a critical factor, as online gameplay is highly sensitive to network speed, which can significantly impact the user experience. To mitigate this, the application must optimize data transfer processes and minimize latency to ensure a smooth gameplay experience. Second, browser compatibility is essential, as the game needs to function seamlessly across various web browsers and devices. This necessitates extensive testing to guarantee that the application adheres to web standards and operates effectively on different platforms. Additionally, security is a paramount concern, particularly with multiple users interacting online. Therefore, implementing strong encryption and secure data handling practices is crucial to safeguard user data. Lastly, the game must be designed responsively to ensure accessibility on various screen sizes and devices, which requires a flexible design approach that may leverage responsive web frameworks.

## [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 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.

The UML class diagram above illustrates the relationships and structure of the software components, showcasing seven classes: ProgramDriver, SingletonTester, Entity, GameService, Game, Team, and Player. At the top, the ProgramDriver class contains the main() method, serving as the entry point for the application. It utilizes the SingletonTester class, which has a testSingleton() method, indicated by the <<uses>> relationship. This design choice suggests that ProgramDriver is responsible for executing the singleton pattern for managing instances of the GameService class, ensuring a single, globally accessible instance.

The Entity class is the base class for Game, Team, and Player, demonstrating the principle of inheritance. Each derived class inherits common attributes such as id and name, along with methods for getting those attributes and converting the object to a string representation. This promotes code reusability and maintains a clean structure by centralizing common functionalities.

The GameService class manages a collection of Game objects and utilizes lists to handle multiple instances efficiently. It has methods for adding games, retrieving game information, and generating unique identifiers for players and teams. The relationship between GameService and Game is represented by a "0...\*" connection, indicating that one GameService can manage multiple Game instances. Similarly, each Game can contain multiple Team objects, which in turn can include multiple Player instances, as shown by their respective connections.

## [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 is known for its great performance and stability, making it suitable for hosting web applications. However, it may be limited in scalability compared to Linux or Windows servers, and the costs of hardware can be higher. | Linux offers excellent performance, security, and scalability for web applications, with lower hosting costs. Its open-source nature allows for customization, but it may require more technical expertise to manage and configure. | Windows provides a user-friendly interface and a wide range of development tools. It's well-integrated with various enterprise applications, but licensing costs can be high, and it may not perform as well under heavy load compared to Linux. | Ordinarily, mobile devices are not used for hosting web applications due to limitations in processing power and resources. However, they can host lightweight apps. The advantage is accessibility; users can access applications from anywhere, but performance can be inconsistent. |
| **Client Side** | Developing for Mac may require specialized skills in macOS development. The costs can be higher due to hardware requirements, and development tools are often more expensive. | Linux development can be more cost-effective, as it utilizes open-source tools. However, it may require a higher level of expertise, and the variety of distributions can complicate support. | Windows offers a broad range of development tools, making it accessible for developers. However, challenges include licensing costs and compatibility issues with older systems. | Mobile app development requires expertise in specific languages (e.g., Swift for iOS, Java/Kotlin for Android) and can be costly in terms of development time and resources. The mobile platform also introduces device fragmentation. |
| **Development Tools** | Common languages include Swift and Objective-C, with tools like Xcode for development. Web frameworks like React or Angular can also be used. | Languages such as Python, Ruby, and PHP are popular, along with IDEs like Eclipse or Visual Studio Code. Tools like Docker can aid in deployment and management. | C#, ASP.NET, and JavaScript are commonly used, with Visual Studio as the primary IDE. Windows also supports a range of frameworks like .NET for web development. | Development tools include frameworks like React Native or Flutter for cross-platform development, and native development tools like Xcode (iOS) and Android Studio (Android). Languages used vary by platform but often include JavaScript, Swift, and Kotlin. |

## Recommendations

1. **Operating Platform**:

Based on the needs of The Gaming Room to expand "Draw It or Lose It" into various computing environments, I recommend using Linux as the primary operating platform. Linux offers excellent performance, scalability, and cost-effectiveness, making it an ideal platform for hosting distributed web applications. Additionally, its open-source nature allows for high levels of customization and integration with cloud services, which can support the distributed environment the game requires.

1. **Operating Systems Architectures**

The Linux operating system employs a monolithic kernel architecture, which means that the core services such as memory management, process management, and networking are handled in one large process. This architecture provides high performance and efficient resource management, which is critical for a real-time multiplayer game like "Draw It or Lose It." Linux’s architecture is also highly modular, allowing for the customization of the system to meet specific needs such as load balancing and security enhancements.

1. **Storage Management**

I recommend using Amazon S3 cloud-based storage solution in conjunction with the Linux platform. This cloud storage will allow for flexible, scalable, and cost-efficient management of game data, player profiles, and game states. It also provides redundancy and high availability, ensuring that the game can handle high traffic and player volume without data loss or downtime.

1. **Memory Management**

Linux uses several memory management techniques, including paging and swapping, which allow efficient utilization of memory resources. These techniques ensure that memory is allocated dynamically, which is crucial for a game that will have varying numbers of players and teams at any given time. Additionally, the Linux kernel's Virtual Memory System can allocate memory to the game efficiently, helping to prevent memory leaks and ensuring smooth gameplay even under heavy loads.

1. **Distributed Systems and Networks**

To support communication between various platforms, "Draw It or Lose It" can utilize RESTful APIs protocols for real-time data exchange. A distributed system could be implemented using microservices architecture, where different components of the game, for instance, game state management and user authentication, are deployed independently across multiple servers. This setup will allow the game to handle connectivity issues like network outages more efficiently, as individual services can be scaled or restarted independently without affecting the entire system.

1. **Security**

Security is quite important for game users. I would recommend the use of Transport Layer Security (TLS) for data in transit and Advanced Encryption Standard (AES) for data at rest for the protection of user information across platforms. Additionally, employing multi-factor authentication (MFA) for user login and role-based access control (RBAC) for managing user privileges will ensure the security of user data. For the Linux platform specifically, SELinux (Security-Enhanced Linux) can be implemented to enforce strict access controls, further securing the game from unauthorized access.