

Draw It or Lose It Web Application

# **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 | 05/23/2025  6/05/2025  6/19/2025 | Sylvia Davis | Completed software design document for Draw It or Lose It.  Updated. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is expanding its Android game, Draw It or Lose It, into a web-based version to support multiple platforms. This design introduces object-oriented strategies and standard software patterns, such as the singleton and iterator, to manage unique game instances, enforce name uniqueness, and allow scalable, multi-user participation.

## Requirements

- Only one game instance may exist in memory at a time  
- Game, team, and player names must be unique  
- Games should allow multiple teams, and teams should support multiple players  
- The solution must be scalable and work across web-based distributed environments

## [Design Constraints](#_2et92p0)

GameService must implement the singleton pattern to maintain only one active instance, ensuring consistency across sessions. Iterator logic is used to check name uniqueness across Game, Team, and Player entities. Additionally, the system must support scalable architecture to function efficiently in a distributed web-based environment with multiple users.

## [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 model shows that Game, Team, and Player classes inherit from a base Entity class. This applies the inheritance principle. Encapsulation is demonstrated through private attributes and public methods. GameService uses the singleton pattern to manage centralized game logic. Iterator patterns are used for name uniqueness validation. These design decisions ensure scalability, reuse, and system stability.

**"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)

In this section, the characteristics, advantages, and limitations of each traditional operating platform—Linux, Mac, and Windows—as well as mobile platforms will be evaluated to help The Gaming Room determine the most effective deployment strategies for their Draw It or Lose It game application. The evaluation considers both server-side and client-side requirements, including compatibility with web-based deployment, development tools, licensing costs, and scalability. Each platform's potential to support a responsive HTML interface across multiple browsers and devices is also assessed. These insights are intended to help guide future development in a distributed environment, ensuring the game performs consistently across all intended platforms.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | While macOS is an excellent platform for client-side development, it is not commonly used as a server platform for hosting web-based applications. Although it shares a UNIX foundation with Linux and provides a stable development environment, macOS lacks native support for scalable production environments. Hosting on macOS may incur higher licensing costs and is rarely used for deployment in enterprise-level web hosting solutions. | Linux is widely regarded as the best platform for hosting web-based applications in server environments. It is open-source, secure, lightweight, and offers excellent scalability. Linux integrates well with modern DevOps tools like Docker and Kubernetes, making it ideal for distributed environments and cloud hosting. Additionally, it has no licensing fees, which is advantageous for companies seeking cost-effective scaling. | Windows provides a strong server-side environment, particularly in enterprise settings. It supports technologies such as ASP.NET and is compatible with many commercial software packages. However, it often incurs higher licensing costs, and system resources may be heavier compared to Linux. It is a good option when integration with other Microsoft services is required but is less preferred for lightweight or open-source hosting environments. | Mobile devices will function as clients. The application must be responsive and accessible on both Android and iOS devices. Although development will not be done on mobile operating systems, user experience must be optimized for mobile access. Windows provides a strong server-side environment, particularly in enterprise settings. It supports technologies such as ASP.NET and is compatible with many commercial software packages. However, it often incurs higher licensing costs, and system resources may be heavier compared to Linux. It is a good option when integration with other Microsoft services is required but is less preferred for lightweight or open-source hosting environments. |
| **Client Side** | Mac devices provide a polished client-side experience, particularly for users who value visual design and interface fluidity. Web applications must support Safari, the default browser on macOS, while also ensuring compatibility with Chrome and Firefox for broader access. Developers should prioritize cross-browser testing to ensure consistent performance and appearance across platforms. Although native Mac desktop apps are an option, using a responsive web interface provides the most flexible and scalable solution. | Linux is commonly used by technically inclined users and developers, making it an important platform for client-side access. Users typically access applications via web browsers like Firefox or Chrome. To ensure compatibility, applications should follow open web standards and avoid proprietary technologies. Since Linux desktops can vary in configuration and performance, testing the web application in different environments is key to ensuring reliable performance and usability. | Windows is the most widely used desktop operating system, making it a critical platform for client-side compatibility. The web application must function well in browsers like Microsoft Edge, Chrome, and Firefox. Windows users expect smooth integration with desktop features, so responsiveness, load speed, and layout flexibility are essential. Supporting accessibility features and varying screen resolutions is also important for reaching a broad user base. | Mobile devices serve as the primary access point for many users, so the application must be designed with mobile-first principles. This includes a touch-friendly interface, fast loading times, and compatibility with both Android and iOS browsers. Developers should use responsive web design techniques to adapt to different screen sizes and orientations. Performance optimization |
| **Development Tools** | Mac development is supported through tools like Xcode for native iOS development and IntelliJ or Eclipse for cross-platform work. macOS supports popular development languages including Swift, Java, Python, and JavaScript. It is well-suited for front-end development thanks to its built-in UNIX terminal and high compatibility with web development tools. Developers can leverage macOS for both browser-based testing and lightweight full-stack development. | Linux supports a wide variety of development tools, including Eclipse, Visual Studio Code, and terminal-based utilities. It is a preferred environment for back-end development and scripting tasks due to its flexibility and open-source nature. Developers commonly use Java, Python, PHP, and JavaScript. Linux also supports DevOps tools, making it ideal for managing deployments, scripting automation, and containerized development with Docker or Kubernetes. | Windows developers often use tools like Visual Studio, Eclipse, and IntelliJ, depending on the project requirements. It supports multiple languages such as C#, Java, Python, and JavaScript. Although Windows is not traditionally used for server hosting, it is well-suited for client-side development, enterprise software, and game engines. Licensing for some tools may be a consideration, but Windows provides a robust environment for testing and debugging web applications. | Mobile development relies on browser-based interaction, so tools like Chrome Developer Tools, Safari’s Web Inspector, and mobile emulators are essential. Developers must use responsive frameworks such as React, Bootstrap, or Vue.js to ensure the application works smoothly across all devices. Android Studio and Xcode are also used for native app testing if needed, but the primary focus remains on optimizing the web interface for mobile performance and user experience. |

## Recommendations Operating Platform: Linux is the recommended operating platform because it offers scalability, strong community support, and compatibility with various web technologies. Its open-source nature and efficient resource utilization make it ideal for deploying Draw It or Lose It across multiple environments, including cloud servers. Operating Systems Architectures: Linux supports a layered and modular architecture that separates user applications, kernel functions, and hardware drivers. It allows developers to build and maintain isolated services using containers, making it highly adaptable for web-based applications like Draw It or Lose It. Storage Management: A Linux-based system can utilize modern file systems like ext4 or XFS, with scalable storage solutions provided through cloud services such as Amazon S3 or Azure Blob Storage. These systems support high availability, redundancy, and secure access for game assets and user data. Memory Management: Linux uses virtual memory and paging to efficiently allocate memory to processes, preventing memory leaks and ensuring smooth multitasking. Draw It or Lose It can benefit from memory isolation between services using containers and cgroups to manage resources per instance. Distributed Systems and Networks: Distributed communication can be achieved using RESTful APIs over HTTPS, allowing clients from various platforms to interact with the server. The use of load balancers and microservices allows services to scale independently. Failover mechanisms and replicated services will minimize the impact of outages or connectivity issues. Security: To protect user data, encryption should be applied both in transit (via HTTPS) and at rest (using encrypted databases). Authentication systems like OAuth2 and token-based access control will restrict access. Linux supports secure firewalls (iptables) and monitoring tools to detect and prevent intrusions, ensuring cross-platform security integrity.

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**: Based on the evaluation of various platforms and system architecture considerations, Linux is recommended as the ideal operating platform for deploying the Draw It or Lose It application. It is scalable, open-source, and integrates well with modern web technologies and development practices. Linux’s strong community support, cost-effectiveness, and performance under high-demand conditions make it a solid choice for server deployment in both on-premise and cloud-based environments.

The layered and modular nature of Linux architecture allows for clear separation between system processes, kernel functions, and user-level applications. This is beneficial when building scalable and maintainable applications, especially when using containers such as Docker for microservices deployment.

1. **Operating Systems Architectures**: In terms of storage management, Linux supports reliable file systems like ext4 and scalable storage solutions through integrations with cloud services such as AWS S3 or Azure Blob Storage. These options provide redundancy, high availability, and secure access for user data and game assets.
2. **Storage Management**: A Linux-based system can utilize modern file systems like ext4 or XFS, wit

h scalable storage solutions provided through cloud services such as Amazon S3 or Azure Blob Storage. These systems support high availability, redundancy, and secure access for game assets and user data.

1. **Memory Management**: Linux also offers strong memory management through the use of virtual memory and paging. Resource management features like control groups (cgroups) ensure efficient memory allocation between different services, which is essential when deploying multiple instances of the application.
2. **Distributed Systems and Networks**: The application can support distributed communication through secure RESTful APIs over HTTPS. Using load balancers and microservices will allow different services to scale independently and support high availability. Implementing failover strategies and redundant services will minimize downtime and maintain reliable access across platforms.
3. **Security**: Security is also a major strength of Linux systems. HTTPS encryption ensures secure data transmission, while encrypted databases protect information at rest. Role-based access control and token-based authentication (such as OAuth2) help maintain user privacy. In addition, firewall configuration, system monitoring tools, and intrusion detection systems can be used to safeguard the application from external threats, supporting a secure and stable environment for all users.