

# Draw It or Lose It Web-Based 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 | 11/24/2024 | Cameron Chenault | Initial draft of design document |

**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 looks to expand their existing Android-only game, *Draw It or Lose It*, into a web-based application capable of serving multiple platforms. This expansion will allow wider accessibility while retaining core gameplay. The project involves adapting the current game into a scalable, distributed system that meets requirements for unique identifiers, team/player management, and single-instance memory control. Using modern software design principles, including object-oriented programming and established design patterns, we propose a robust architecture that ensures flexibility, maintainability, and efficient resource utilization.

## Requirements

*The primary requirements for the application include the support of one or more teams, each with multiple players. The enforcement of unique games, teams, and player names. A single game instance in memory at any given time, ensuring data integrity and consistency. And assignment of unique identifiers for all entities (game, team, and player).*

## [Design Constraints](#_2et92p0)

Designing a web-based, distributed environment introduces constraints like Concurrency, having multiple users simultaneously and necessitating synchronized data management. Scalability**,** the application will have to support varying user loads across platforms without a drop in performance.

Cross-Platform Compatibility**,** having a unified codebase catered to different operating systems and devices. Security, sensitive user data must be protected through encryption and secure authentication methods.

## [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 diagram outlines relationships between the primary classes: Game, Team, and Player, all derived from a base Entity class containing shared attributes like id and name. Key object-oriented principles employed include encapsulation, inheritance, and polymorphism. Entity ensures all attributes and behaviors are centralized and reusable. Game, Team, and Player inherit common properties, reducing redundancy. Unique implementations for addTeam() and addPlayer() ensure flexibility while maintaining a standard interface. These principles support maintainability, efficiency, and adherence to software requirements.

**"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** | MacOS uses a Unix based environment, making it suitable for hosting web-based applications. macOS is less commonly used as a server OS and has a high licensing cost. MacOS is known to be stability but has little large scale server deployment use cases. | Linux is widely regarded as the best option for web hosting due to its open-source nature, high performance, and scalability. It is more cost effective for clients due to low to no licensing cost. | Windows offers a user friendly interface and broad compatibility with large scale or enterprise software, but licensing can be expensive. It is highly scalable but does requires more powerful hardware. | Mobile devices are not used as web servers. Though they must interface effectively with the server application. |
| **Client Side** | Developing for macOS requires knowledge of Swift and Objective-C for compatibility. High hardware costs for Mac systems. Extra time may be needed for UI/UX refinement to meet user expectations. Development time may increase if the team is unfamiliar with Apple’s ecosystem. | Developing for Linux clients involves creating cross-platform web applications that run on browsers. Compatibility is key for Linux due to its open-source environment. There is minimal cost involved. Expertise in debugging and testing on Linux distributions may require more time. | Development for Windows requires combability with major browsers. Expertise and licensing costs for Windows specific tools like Visual Studio should be considered. Windows is widely used so significant time will go toward optimizing for accessibility and performance. | Mobile development for iOS and Android requires expertise in frameworks like Swift for iOS and Java for Android. Time will play a major factor due to the need for a seamless experience across a variety of different OS versions and screen sizes. Cross platform solutions like React can save time but may require specialized expertise. |
| **Development Tools** | macOS developers can use tools like Xcode, Swift, or Visual Studio Code. Swift is the native language for Apple development, but additional tools may require licensing fees. | Linux supports a wide array of development tools, such as Visual Studio Code, and frameworks like Django or Node.js. All are free and open source. | Windows supports a wide range of development environments, including Visual Studio. Additional costs may occur due to licensing for some tools. | Mobile development for iOS and Android often requires separate tools, like Xcode for iOS and Android Studio for Android. |

## 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**: The operating platform that I would recommend being using for this game is Linux. Linux is widely used for server-based environments and would allow for play on other operating platforms. Plus, it has traits that make it very desirable such as cost effectiveness, be open source, and its scalability.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel architecture. This architecture integrates services like file management, process scheduling, and memory management directly into the kernel. This makes Linux highly efficient system with low overhead, ideal for applications demanding high performance and stability. Linux’s kernel is also modular, this means specific modules of the kernel can be loaded and unloaded based off the need of the project at hand. This gives the serve a lot of flexibility when managing other resources.
3. **Storage Management**: The Fourth Extended Filesystem or “Ext4” is recommended for storage management system for a Linux base platform. Ext4 ensures data integrity by maintaining a record of changes before they are committed to the main file system. This is important for handling game data, such as player progress and game states. For backup and disaster recovery, integrating cloud-based storage solutions, such as AWS S3, will provide redundancy and ensure data availability.
4. **Memory Management**: Linux has advanced memory management techniques like virtual memory. This allows the system to allocate memory dynamically and efficiently even under a high load of user. For this memory management will prioritize keeping the game’s current state in memory while offloading less frequently accessed data to secondary storage.
5. **Distributed Systems and Networks**: To enable seamless communication across various platforms a distributed system using a client-server is recommended. Game states and player interactions will be managed through a server that will act as a central hub for this data. It is important maintain low latency and to allow real-time data exchange for a multiplayer game. To ensure connectivity and mitigate issues like network outages, implementing load balancers and redundant servers in geographically distributed data centers will improve reliability and uptime
6. **Security**: Linux has very strong security features that will ensure used data protection as well as system wide protection. These features include SELinux which handles access control, iptables which handles network security, TLS for data encryption, and OAuth for authentication. The sum of these features will ensure Draw It or Lose it stays secure while in operation.