

**Draw It or Lose It**

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 10/19/25 | Samantha Vaillancourt | Recommendations |

**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)

Our goal is to design and implement a web-based version of “The Gaming Room’s” existing Android game, *Draw It or Lose It*. The objective is to extend the game’s reach by making it accessible across multiple platforms, including web browsers, desktop operating systems (OS), and mobile devices, while preserving the fun and competitive spirit of the original app.

The new OS will introduce scalability, accessibility, and cross-platform compatibility. By leveraging a web-based distributed environment, the game will allow multiple teams to compete in real time, with unique game and team identifiers to avoid duplication. Additionally, enforcing the rule that only one instance of a game may exist in memory at any time will ensure consistency, prevent data conflicts, and maintain game integrity.

It is recommended to develop a centralized, server-hosted application that uses current web technologies for the client interface and secure backend services for game state management. This will provide the foundation to scale as demand increases while meeting the client’s functional requirements.

## Requirements

*Business: Expand the existing game from an android-only app to a web-based program through multi-platform accessibility, ensure an attractive and competitive multiplayer experience over multiple devices, provide simple setup for entry level teams and players, maintain integrity and game rule consistency with the original android version.*

*Technical: OS must support one or more teams per game, each team must support multiple players, game and team names cannot repeat, only one game instance can exist in memory at one time, and the application should run in a distributed web-based environment to allow access from multiple platforms.*

## [Design Constraints](#_2et92p0)

Cross-platform support: The application must run smoothly on desktops and mobile devices, requiring responsive web technologies to avoid multiple codebases.

Unique identifiers: Each game, team, and player must have unique IDs, enforced by the backend coding to prevent duplication.

Single instance rule: Only one game instance can exist in memory at a time, requiring careful server session and lock management.

Scalability: The system must handle many concurrent users, which calls for distributed state management, caching, and load balancing.

Security: Real-time play demands secure communication, authentication, and data integrity.

Cost efficiency: The solution should leverage cloud services and open-source tools to balance performance with budget considerations.

## [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)

Entity is the base class that now creates a relationship between Game, Team, and Player as subclasses of Entity. The subclasses will inherit items from the parent (super) class, Entity. GameService is a singleton class, but manages all game instances. ProgramDriver is the main() method and SingletonTester assures that only one GameService instance exists. The best part of a UML is that it shows the inheritance of classes easily. We can also point of that ProgramDriver depends on SingletonTester which in turn uses GameService. For OOP, we see encapsulation as each class defines its own attributes and behaviors, inheritance which we covered regarding the Entity class, polymorphism where inherited methods like toString() can be overridden in subclasses for other customized behavior, abstraction by separating the parent and subclasses which makes the system easier to maintain and extend, and singleton pattern in GameService that assures only one service instance exists in memory.

**"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 are not typical for industry standard for large-scale server hosting, but are unix-based and user friendly. The macOS server advantages are the strong development environment, top-notch security, and reliability. The weaknesses are higher costs for software, and the uncommon OS in most IT careers. | Most predominantly used OS because of its open-source advantage, easily customizable, strong security, scalable, and cost effective. The weakness would be the user to have more technical expertise to navigate the OS, and there are fragmented distributions. | Windows is widely used, almost as much as Linux, but more geared towards enterprise environments. It has a strong support for .NET applications and integrates easily with Microsoft services. Its weaknesses are the costs, data usage, and lacks security compared to mac and Linux. | Not typically used as a supportive device for an OS. Advantages are that mobile devicdes are lightweight, portable, and app friendly, however they have little processing power and inability to maintain a long battery life, and not scalable for production servers. |
| **Client Side** | User-friendly and one of the most widely used OS for general use. Development for its users requires compatibility with macOS interface guidelines. Costs for mac hardware and software is pricier but with HTML interface, overall cost would be lower for the application or web access. Limited areas of expertise compatible with macOS; apple-specific languages include Swift and Objective-C. | Where Linux requires more technical expertise, it can be complex for the user due to the different distributions. Consider the open-source tools, extra time for compatibility testing and review, less OS user base, and developer expertise geared towards a graphics interface. Where access to application would be through the web, costs would remain low. | The most preferred by clients for desktop usage. Consider the amount of users who prefer Windows, wide array of hardware variability and programming tools. Cost is average and minimum expertise is required. Again, where this a client would access the application through the web, overall cost would be low. | Predominant for user access for flexibility. Consider platform differences (apple vs android) and frequent OS updates. It can become costly to operate multiple systems, and requires cross-platform programming for semi-compatibility. Specific expertise is required. The responsive HTML design is essential in granting IOS users access. Both Android and IOS mobile users can use the application through their mobile device browser. |
| **Development Tools** | Programming languages and tools: Swift, Objective-C, and macOS software development kits. Compatible with development tools for cross-platforming like Visual Studios, PyCharm, and GitHub accessory. Developers will need experience with Apple’s development ecosystem and potentially establish a separate team to take over the cross-platforming criteria depending on the depth of the application. MacOS offers Xcode for free. | Programming languages and tools: C, C++, Java, Python, Node.js, Ruby, PHP, and Eclipse. The benefit to Linux is it has a strong open-source system allowing compatibility with a wide array of languages which also makes it cost-efficient. Linux’s open-source does not require licensing fees which makes this OS so budget-friendly for larger teams/projects. | Programming languages and tools: C#, C++, .NET, Java, Python, Visual Studio, and PowerShell. Windows typically requires .NET expertise, but VS integrates well for enterprise environments. However, Microsoft has its own challenges with its systems and may require a designated development team. VS is also free for smaller projects, but Professional and Enterprise editions require a license. | Programming languages and tools: Swift for iOS, Kotlin for Android, and cross-platform IDE’s like Flutter and Unity. The cross-platform tools allow code reuse which reduces the need to make subteams. There are a few free programming languages like Flutter and Swift, but being on a mobile device, Apple and Google Play will typically charge. |

## 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**: My first recommendation would be Linux as it is the most stable and cost effective for hosting web-based, multiplatform applications. It has the strongest compatibility with Python, Java, and C#. Linux provides autonomous services to decrease complexity and provide consistent performance across computing environments. It is supported in many server environments, open-source, and scalable.
2. **Operating Systems Architectures**: Linux is preferred (Microsoft is a close second) because of its memory management and support for large architecture. Linux is also great for multiuser OS and has strong networking support and the preferred operating platform for graphic interface. It is the most effective for multitasking, processing isolation, and resource control. Linux is also customizable and secure for server hosting.
3. **Storage Management**: Linux is the best choice for hosting using a centralized cloud-based storage system because of its capacity and distributed storage systems. Along with its scalability and security. A relational database is able to store data, game states, and authentication details which are compatible with all major server operating systems. Using a cloud-based system automates cloud backups and replication ensures data persistence and quick recovery in case of failure or compromise.
4. **Memory Management**: Mac is very reliable for memory management, especially for applications, but limited for server-scale hosting. In this circumstance, I would choose Linux again. It can isolate processes and reclaim unused memories. It provides partial memory distribution by storing and distributing through virtual memory and demand paging. It has efficient virtual memory and would work great on the backend to enforce the one-game instance in memory.
5. **Distributed Systems and Networks**: Linux hosts distributed services, but Mac/Windows/mobile devices act as client platforms connecting to the backend. This would be where cross-platform accessibility would come into play. Scalability must handle multiple users over different operating systems and balance overload, so serverless architecture is the best way to support that growth. Managing growth with a properly distributed system, like Linux, will help stream continuous connectivity.
6. **Security**: Either Linux or Windows is applicable for security. Linux has a strong security model with firewalls, ideal for securing web services and preferred backends. Windows offers user authentication features, but less efficient for lightweight game hosting. To start the game, authentication is necessary for login credentials. This allows individuality and security for each player, and advanced access for admins in charge of the application’s functionality. Throughout the course of the game, monitoring and updates help safeguard against evolving threats by detecting intrusion, firewall configuration, and monitoring tools