

Draw It or Lose It

# **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 | 05/21/2025 | Albert Waterman | Included Executive Summary, Design Constraints, and Domain Model sections based on client requirements and provided UML diagram. |
| 1.1 | 06/06/2025 | Albert Waterman | Update to evaluation. |
| 1.2 | 06/22/205 | Albert Waterman | Update Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

Gaming Room is expanding its Android game, Draw It or Lose It, into a web-based, cross-platform version. The goal is to develop a scalable, maintainable game architecture that is capable of supporting multiple teams, containing multiple players, in a four-round timed drawing game. To meet the clients needs I’m proposing a Java-based application that uses object-oriented programming and the Singleton design pattern to make sure there’s only one instance of the game service running. This setup makes it easy to keep track of games, teams, and players by giving each one a unique ID and making sure names aren’t duplicated. The UML diagram lays out a solid structure that’s easy to build on and flexible enough to support future updates, like expanding to more platforms or adding web integration.

## Requirements

Business Requirements:

-expanding the current game from not only Android, but to a web-based platform that can support multiple devices.

-Include the same gameplay structure

-Allow teams to be created and join teams with unique identifiers.

-improve scalability and user management

Technical Requirements:

-Using the singleton pattern, only one instance of the game should exist in memory

-supports multiple teams per game and multiple players per team

-Use unique names for games and teams to prevent duplication

-Assign (ID) to every game, team, and player.

-Use object-oriented programming to organize code efficiently.

## [Design Constraints](#_2et92p0)

Developing the game in a web-based environment comes with several important constraints. The system must handle multiple users simultaneously, which requires efficient session management and secure communication between the client and server. Since the game runs in real-time with timed rounds and team interactions, it’s essential to minimize lag and handle multiple activities smoothly. Only one instance of the game service should exist in memory, so the Singleton pattern is used to keep control centralized. Games, teams, and players must each have unique names and IDs, which requires adding validation checks, especially since users may be accessing the system at the same time. The app also needs to be built in a way that allows for easy updates later, like adding login features or connecting to a database, without having to rebuild the whole thing.

## [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 the core structure of the gaming application and shows how the system's components are organized using object-oriented principles. The center of the model has the Entity class, which is the base class for Game, Team, and Player. This uses inheritance with shared attributes like id and name to be reused with multiple classes. The game holds Team objects and then each team contains a list of player objects. This association shows a one-to-many relationship, where a game can have multiple teams, and each team has multiple players. This aligns with the client's requirement for multiple team and player management within a single game session. The GameService class uses a singleton, showing that one instance of the game service exists in memory. It also has a unique ID for games, teams, and players, with methods for creating and retrieving these entities. Encapsulation is shown through private fields and public methods which is used to protect the internal state of objects and secure interaction with class data. Polymorphism is used with toString() method in each class.

**"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 can technically host a web-based application, but it’s not really built for that role long-term. It’s secure and reliable since it’s Unix-based, but most people don’t use it for running live production servers. It works better for development or testing than full-scale hosting. | Linux is one of the best options for hosting web apps. It’s free, super stable, and secure. It supports all the major tools and servers like Apache, Nginx, and Node.js. It’s flexible and has a huge community, which helps with troubleshooting and long-term support. | Windows is used a lot in businesses for hosting apps, especially if they use Microsoft tech like .NET or SQL Server. It’s solid and easy to use, but it costs more because of licenses. It also needs regular updates and maintenance to stay secure. | Phones and tablets don’t host web apps, but they need to connect to them. That means your server must be mobile-friendly, fast, responsive, and secure. Everything should load quickly and work on smaller screens. |
| **Client Side** | When supporting Mac users, development can take more time and cost more since there aren’t as many tools that cover all platforms. You’ll also need people who understand macOS well to make sure everything runs smoothly and feels native to Mac users | Linux isn’t as common for regular users, but it works well for certain clients, like developers or people using open-source tools. It’s cheap to develop for, but you’ll need someone who knows the ins and outs of Linux systems to get everything working right. | Most users are on Windows, so supporting it is a must. It’s easier to develop for since there’s tons of support and tools available. Just make sure to test on other systems too if you’re going cross-platform. | Mobile app development can get expensive and time-consuming, especially if you’re building for both iOS and Android. You either have to make two versions or use a cross-platform tool. You also need to keep up with updates and app store rules. |
| **Development Tools** | Developers on Mac usually use Xcode, IntelliJ, or VS Code. Swift and Objective-C are common for Apple-based development, but you can also use Java or Python. Xcode is beneficial if you’re planning to go into iOS later too. | Linux works with a lot of programming tools like VS Code, Eclipse, and IntelliJ. It supports Java, Python, C++, and more. You can also use the terminal and shell scripting to automate tasks, which makes it great for backend and server development. | Visual Studio is the go-to for Windows .NET, PowerShell, and even Java and Python are common languages. Windows has everything built in to support app development, especially for desktop and enterprise software. | Most Android apps are made with Android Studio using Java or Kotlin. iOS apps are built in Xcode using Swift or Objective-C. If you want to build one app for both platforms, tools like Flutter or React Native are great options and save a lot of time. |

## 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**: Windows is my best choice because it's very user-friendly, integrates well with .NET applications, and is used a lot in business environments. It offers good support for tools, making it a reliable option for scaling the game across multiple platforms.
2. **Operating Systems Architectures**: Windows uses a hybrid kernel architecture using both microkernel and monolithic designs. This allows system stability and flexibility by separating services and keeping performance high. Windows also uses applications like file management and a graphical user interface.
3. **Storage Management**: Windows supports NTFS which is very secure, scalable, and has built in support for access control, encryption, and file compression like zip files. So tools like storage spaces have volume management, which will help with backing up game data and scaling larger storage as the game increases in size.
4. **Memory Management**: Windows uses a virtual memory system with paging and prioritization. It keeps active processes in RAM while pushing fewer active ones to the page file. This helps manage system performance during peak gameplay, ensuring that timed rounds and real-time interactions stay smooth even when the load increases.
5. **Distributed Systems and Networks**: For handling communication across platforms, I’d recommend using RESTful APIs connected to a backend hosted on a Windows Server. This setup allows the game to support web and mobile access while keeping the data flow organized and secure. Since the client expects the game to scale and perform well under load, a dedicated server or cloud solution, such as Microsoft Azure, would be ideal. It offers better control over performance, uptime, and resource allocation compared to shared options, which is important for real-time games like this.
6. **Security**: Windows Server has built-in security tools like Windows Defender, BitLocker, and firewall services. It supports role-based access control (RBAC), encryption, and secure authentication methods like Active Directory and multifactor authentication. Regular patching and automated updates help reduce vulnerabilities and keep user data safe across all platforms.