

# **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 | 04/18/25 | Lawrence Langelier | Final update for Project Three submission. Recommendations section completed with details on OS architecture, storage, memory, distributed systems, and security. |

**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)

To expand *Draw It or Lose It* into a web-based, cross-platform game, we propose a design that uses object-oriented principles and a singleton pattern to manage a single game instance. Each game, team, and player will have a unique ID, and names will be checked for uniqueness. This approach ensures the application is organized, scalable, and ready for development.

## Requirements

*  The game must support one or more teams per match.
*  Each team must allow multiple players.
*  Game and team names must be unique to avoid duplication.
*  Only one instance of the game should exist in memory at a time.
*  Each game, team, and player must have a unique identifier.
*  The application must be web-based and function across multiple platforms (desktop and mobile).

## [Design Constraints](#_2et92p0)

*  **Single Game Instance**: Only one game instance can exist in memory, requiring the use of the singleton design pattern to manage it.
*  **Unique Names and IDs**: All game, team, and player names must be unique. This requires validation and efficient storage/retrieval methods to prevent duplication.
*  **Cross-Platform Support**: The game must work across multiple platforms, so the design must use web technologies that are platform-independent.
*  **Scalability and Concurrency**: With multiple teams and players active at once, the system must manage concurrent access to shared resources without conflicts.
*  **Web-Based Environment**: Being browser-accessible means managing sessions, latency, and user state without relying on local storage or native mobile features.

## [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 uses object-oriented principles to model the game structure. Game, Team, and Player inherit from the Entity class, which provides shared id and name attributes. A Game holds teams, and each Team holds players, showing composition. The GameService class uses the singleton pattern to ensure only one game instance exists in memory. This design keeps the system organized, supports unique identifiers, and meets the client’s key requirements efficiently.

**"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 host web apps, but it's less commonly used in production due to limited server-grade tools and higher hardware costs. | Linux is ideal for hosting; it's stable, open-source, widely supported, and most cloud providers use it. | Windows can host web apps using IIS and supports .NET. It's good for enterprise use but has licensing costs and uses more system resources than Linux. | Windows can host web apps but often comes with licensing costs and can be less efficient than Linux. |
| **Client Side** | Mac users expect sleek UI and strong performance. Development tools like Xcode are available, but fewer users overall. | Linux clients are less common, but browser-based apps will still work fine. Dev support may be limited. | Windows is widely used and must be supported. It offers strong dev tools and broad compatibility. | Must support Android and iOS; most players will access the game from mobile devices, so responsive design is critical. |
| **Development Tools** | Supports tools like Xcode, VS Code, and IntelliJ. Good for frontend testing and cross-platform development. | Strong open-source support. Ideal for backend dev with tools like Eclipse, VS Code, and terminal utilities. | Supports Visual Studio, Unity, and VS Code. Some tools are free, others paid. Works well with C#, Python, and JavaScript. Widely used and easy for dev teams. | Mobile dev requires Android Studio or Xcode. Cross-platform tools like Flutter or React Native can streamline development. |

## 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**: I recommend using a Linux-based server environment with a web-based frontend for cross-platform compatibility. Linux is stable, cost-effective, and widely supported, making it ideal for hosting scalable web applications.
2. **Operating Systems Architectures**: The backend will use a 64-bit Linux architecture, allowing high performance and resource management. The frontend will be accessible via modern browsers on Windows, macOS, and mobile OSes (iOS/Android), supporting client diversity
3. **Storage Management**: I recommend using an online cloud storage system that can grow as the game grows. It keeps game data safe, backed up, and easy to access when needed.
4. **Memory Management**: The Linux server uses virtual memory and process isolation to manage system memory efficiently. With only one game instance in memory at a time (using a singleton pattern), memory usage remains low and controlled.
5. **Distributed Systems and Networks**: The game will use a web connection to let players interact in real time, no matter what device they are using. Hosting it in the cloud keeps it running smoothly, even if one part goes down. This setup helps the game stay online and working during busy times or small outages.
6. **Security**: All user data should be kept safe by using secure web connections so no one can see or steal information while it travels online. The system will check all user input to block anything harmful, and accounts will be protected with secure logins. User information will be stored safely in a protected database. Regular updates will be made to fix any security problems, and extra steps will be taken to keep hackers out and the system secure on all devices.