

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 | 07/14/25 | Lizette Trevino | This is an initial design summary that outlines Draw It or Lose It’s project requirements, design constraints, and rationale for development decisions. |

**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 is a game development company that is looking to expand access to its drawing-based party game, Draw It or Lose It, by creating a web-based version that supports multiple platforms. To accomplish this, the solution must not only meet software requirements but also operate effectively within a scalable and distributed environment. The new platform must ensure compatibility with different operating systems such as Windows, Mac, Linux, and mobile browsers; reliable memory management to prevent multiple game instances from running simultaneously; and secure communication across networks.

The system will support multiple teams and players, enforce unique names for games and teams, and maintain only a single active instance of the game in memory at any given time. In addition, the architecture must support scalability, allowing additional users and teams to be onboarded as the platform grows. Security is a priority, requiring protections for user data, authentication across devices, and secure transmission of information. By addressing both the software design and operating environment requirements, this project provides a reliable, secure, and scalable solution for expanding Draw It or Lose It into a web-based, multiplatform application.

## Requirements

Technical Requirements:

1. The application must function in a web-based environment.
2. The system must support multiple teams and multiple players.
3. Each game and team must have a unique name.
4. Only one instance of the game should exist in memory at any given time.

Business Requirements:

1. The current Android game must be redeveloped as a web-based, multiplatform application.
2. The solution must deliver a consistent and reliable gameplay experience across platforms.
3. The system should be scalable and able to support additional users and teams as the platform grows.

## [Design Constraints](#_2et92p0) 1. Single Instance of the Game:

Only one game session can run at a time. This will be enforced using the Singleton design pattern to manage memory efficiently and prevent duplication of game states.

2. Unique Identifiers for Games and Teams:

Each game and team must have a distinct name. The system will validate and assign unique identifiers to maintain consistency across users and platforms.

3. Concurrency and Multi-User Access:

Multiple users and teams may interact with the system at the same time. The design must handle concurrent access without performance degradation or data conflicts.

4. Cross-Platform Operation:

The application must run consistently across major web browsers on Windows, Mac, Linux, and mobile devices. This requires ensuring compatibility in rendering, networking, and memory handling.

5. Application Architecture Shift:

Unlike the Android-only app, the new version must operate in a client-server environment. This requires scalable server infrastructure to handle multiple users, load balancing for performance, and distributed storage solutions for hosting stock images used in gameplay.

6. Storage and Memory Management:  
Because gameplay relies on rendering a large library of images, the system must use efficient storage (preferably cloud-based) and memory management to deliver fast, reliable performance.

7. Security Across Platforms:  
The transition from a mobile app to a web-based environment introduces new security challenges. Authentication must be enforced across all platforms, data must be encrypted in transit, and user sessions must be secured to protect against unauthorized access.

8. Performance in Distributed Environments:  
The system must maintain reliable performance even under slow connections or heavy user loads. This includes handling outages gracefully, ensuring fault tolerance, and maintaining synchronization across devices.

## [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)

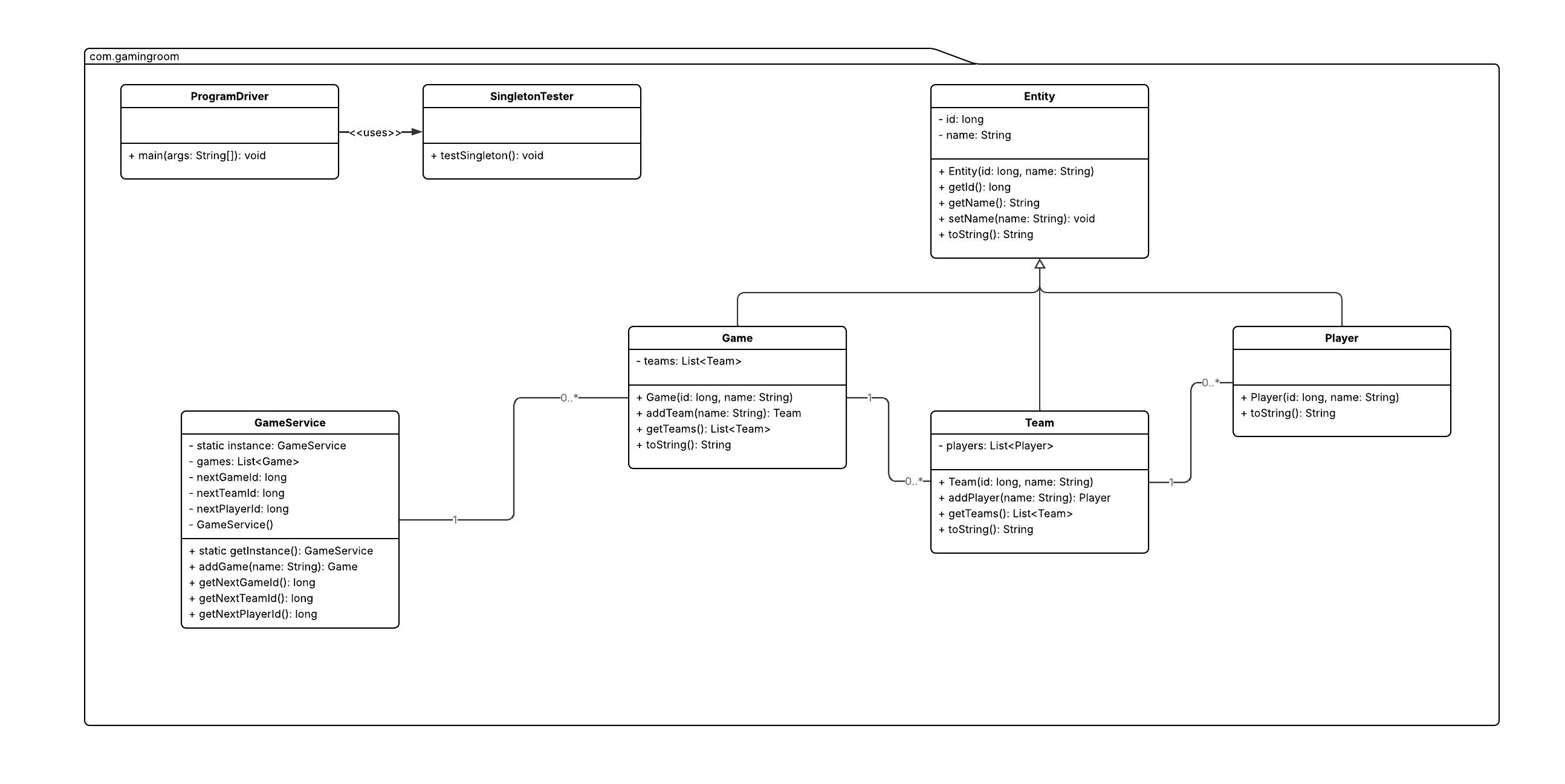
The UML diagram shows how the classes in the game application relate to each other and how object-oriented programming helps organize the system:

* The Entity class is the parent class. It stores common properties such as id and name. The Game, Team, and Player classes all inherit from the Entity class to avoid repeating code.
* The GameService class is the core manager class. It follows the singleton pattern so only one instance is used throughout the program, which helps manage memory usage and performance.
* The Game class contains a list of teams. The addTeam() method allows games to organize different teams.
* The Team class contains a list of players. The addPlayer() method allows teams to include individual players.
* The ProgramDriver class is the main entry point. It uses the SingletonTester class to make sure the singleton pattern is working correctly.

OOP Principles Used:

* Inheritance: Game, Team, and Player all extend Entity.
* Encapsulation: Each class stores and protects its data, using public methods to access it.
* Abstraction: Only the important functions are exposed, like getName() or addGame().
* Single Responsibility: Each class focuses on one task. For example, Game manages teams, and Team manages players.

The Gaming Room UML Diagram



## [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 servers are secure and reliable but are not commonly used for large-scale hosting. They require Apple hardware, which increases costs, and they offer fewer hosting options compared to Linux and Windows. | Linux is the strongest option for hosting web-based applications. It is free, stable, secure, and widely used across the industry. It scales well but has a steeper learning curve. | Windows servers provide a familiar environment with strong development and testing tools. They are well supported but come with higher costs due to licensing fees and greater risk of malware and viruses. | Mobile devices are not suitable for hosting the server side of an application. They are designed to act as clients rather than servers and cannot support the same number of users as traditional operating systems. |
| **Client Side** | Supporting Mac clients involves higher upfront costs due to Apple hardware. Development is done with Xcode and requires knowledge of Swift or Objective-C. Apple has strict validation and review processes, which make development and testing take longer. | Linux clients involve minimal costs since the OS and most tools are free. Testing is slower because of the many distributions available. Developers often need open-source toolchain experience and possibly cross-platform frameworks. | Windows clients involve moderate licensing costs, but development is generally fast due to the wide availability of tools. Many developers already have experience with C# and .NET, making Windows client support easier. | Mobile client development requires publishing to app stores such as Apple App Store or Google Play. Apple charges $99 per year, while Google charges a one-time $25 fee. Testing is slower due to many devices and screen sizes, but mobile platforms reach the widest audience. |
| **Development Tools** | Mac development uses Xcode with Swift and Objective-C. Xcode is free but only runs on Apple hardware, which adds cost. Developers must also be familiar with Apple’s development ecosystem. | Linux development relies on open-source tools like VS Code, Java, Python, and Node.js. These are generally free, but developers must be comfortable with open-source frameworks and packaging systems. | Windows development typically uses Visual Studio with C# and .NET. The community edition is free, but advanced versions may require licensing. Most developers are familiar with Microsoft’s tools, reducing training time. | Mobile development uses Android Studio for Android and Xcode for iOS. Cross-platform tools such as Flutter, React Native, or Unity are also common. Each platform has app store requirements, which add extra steps to development and testing. |

## 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 most effective server platform for this project is Linux. It is cost-efficient, secure, and highly scalable compared to Windows or Mac alternatives. Because Linux is open source, the client avoids expensive licensing fees while still benefiting from strong community support and flexibility. Its stability makes it the standard for web servers and cloud environments, which ensures the Draw It or Lose It game can grow without hitting unnecessary limitations.

1. **Operating Systems Architectures:**

A client–server architecture is the best fit for Draw It or Lose It. The game logic, authentication, and image library will be hosted on the server, while clients connect through a web interface. This setup allows the server to handle the heavy lifting while the clients only need a modern browser. It also ensures a consistent experience across platforms without requiring separate installations or device-specific builds.

1. **Storage Management:**

For storage, a cloud-based solution should be used. The game will need to maintain its library of stock images, store team and player information, and manage session data. A cloud storage system, such as AWS S3 or Google Cloud Storage, is more scalable and secure than on-premises storage. Cloud storage also guarantees availability, meaning the game’s resources can be accessed anytime, even during peak demand.

1. **Memory Management:**

The system should take advantage of Java’s built-in garbage collection to free unused resources and prevent memory leaks. Additionally, the Singleton design pattern will control memory usage by ensuring that only one instance of a game is active at a time. This eliminates duplicate sessions and keeps resource use consistent.

1. **Distributed Systems and Networks:**

Since The Gaming Room wants the game to be playable across multiple devices and platforms, a distributed system is essential. The server should be supported by load balancers and redundancy, so if one server fails, another can immediately take over. This prevents downtime and keeps gameplay uninterrupted. Network communication will rely on secure HTTPS protocols, which protect player interactions and data transfers.

1. **Security:**

Security is a must-have. All player and session data must be protected through encryption in transit and encryption at rest on the server. Authentication should follow modern standards like OAuth2, with optional multi-factor authentication (or MFA) for additional security. Session management should prevent hijacking by enforcing timeouts and regenerating session IDs. The principle of least privilege should be applied so that user accounts and services only have access to what they need, reducing the attack surface.

**Resources**

Silberschatz, A., Galvin, P. B., & Gagne, G. (2008, July). Operating System Concepts, 8th edition. O’Reilly Online Learning. <https://learning.oreilly.com/library/view/operating-system-concepts/9780470128725/>

Ha, D. (2024, July). 9 reasons linux is a popular choice for servers. LogicMonitor. <https://www.logicmonitor.com/blog/9-reasons-linux-is-a-popular-choice-for-servers>

Linux: Why is it the most liked OS for developers?. DataScientest. (2023, February). <https://datascientest.com/en/linux-the-preferred-os-for-developers>

Linux Servers vs. Microsoft Servers: Comparing Differences and Benefits. Volico Data Centers. (2025, July). <https://www.volico.com/linux-servers-vs-microsoft-servers-comparing-differences-and-benefits/>

Panwar, A. (2024, October 28). IOS vs Android App Development: A detailed comparison. Simpalm. <https://www.simpalm.com/blog/ios-vs-android-for-app-development>

Aggarwal, S. (2025, January 10). Mobile App Development Costs in 2025: A Comprehensive Guide. Tech Ahead Corp. <https://www.techaheadcorp.com/blog/mobile-app-development-costs-in-2025-a-comprehensive-guide/>

What is Xcode: Features, Installation, Uses, Advantages and Limitations. BrowserStack. (2025, January 17). <https://www.browserstack.com/guide/what-is-xcode>

Berveno, M. (2025, April 26). 15 best Mac tools for developers: My personal picks. SetApp. <https://setapp.com/lifestyle/best-developer-tools-for-mac?srsltid=AfmBOooh4S8hOKP2v9ZbSHIwFcTTU2LCV5HX5qIDnmxmxHgH0k0uo-0L>

Ekren, E. K. (2025, April 29). What Is Xcode and How to Use It?. NetGuru. <https://www.netguru.com/blog/what-is-xcode-and-how-to-use-it>

LaMear, R. (2025, July 9). More Microsoft Price Increases for Enterprises Take Effect July 1, 2025. US Cloud. <https://www.uscloud.com/blog/more-microsoft-price-increases-for-enterprises-take-effect-july-1-2025/>