

# **Draw It or Lose It Web 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 |
| --- | --- | --- | --- |
| 2.0 | 4/20/2025 | Jake Davis | Final draft including memory/storage management and architecture refinement. |

**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 aims to expand its game, *Draw It or Lose It*, from an Android app to a web-based version that supports multiple platforms.  
This document provides a structured approach to software development, ensuring efficient game management, unique naming, and system stability using object-oriented programming principles and design patterns such as Singleton and Iterator.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

The game application must function in a **web-based distributed environment**, meaning it should be accessible across various platforms such as Windows, Mac, Linux, and mobile devices.  
It must implement a **Singleton pattern** to restrict multiple instances and an **Iterator pattern** to manage game, team, and player uniqueness.  
Additionally, the application must handle real-time player interactions while maintaining a **secure and scalable architecture**.

## [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 class diagram represents the relationships between different entities in the game application.  
The **Entity class** serves as a **base class** with common attributes (id, name).  
The **Game, Team, and Player** classes **inherit from Entity**, ensuring **code reusability**.  
The **Singleton pattern** ensures only **one GameService instance** exists, and the **Iterator pattern** manages **unique names for games and teams**.  
These principles streamline the game’s structure while enforcing **object-oriented best practices**.

**"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 servers** are Unix-based and provide decent performance, but they are **not widely used** in enterprise environments for hosting web applications. MacOS servers are costly and less supported. | **Linux is the most commonly used OS for web servers**. It is free, secure, and widely supported. Apache and Nginx run efficiently on Linux, making it the best choice. | **Windows servers** are an option but require **licenses** and are heavier than Linux. Windows Server is preferred for Microsoft-based web applications using IIS. | Mobile devices **cannot act as primary servers**, but **cloud-based services** like AWS, Firebase, or Azure allow backend functionality to work seamlessly. |
| **Client Side** | **MacOS supports web applications**, but Mac users are a smaller portion of the market. The game will be **accessible via a web browser** with no extra installations required. | **Linux is not widely used as a gaming client, but browsers like Chrome and Firefox allow gameplay. Linux requires additional testing for compatibility.** | **Windows is the most widely used OS for gaming. The game should be optimized for Windows users, ensuring smooth browser-based gameplay.** | **Mobile devices are critical for gaming. The web app must be mobile-responsive and optimized for touchscreen inputs. Android/iOS users will access it via a web browser.** |
| **Development Tools** | Developers can use **Xcode** for mobile development and IntelliJ/Eclipse for Java-based applications. | **Linux is ideal for development** using tools like **Eclipse, VS Code, and MySQL/PostgreSQL** for databases. Open-source libraries are well-supported. | **Windows supports multiple development environments**, including **Visual Studio, Eclipse, IntelliJ, and .NET-based frameworks**. | **Mobile development will use cross-platform tools** like **Flutter, React Native, and Unity**. The backend must support RESTful APIs for web and mobile integration. |

## 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 best operating platform for The Gaming Room to expand *Draw It or Lose It* is **Linux-based cloud hosting**. Linux provides a **stable, scalable, and cost-effective** environment for web-based applications. It is widely supported, secure, and compatible with essential web technologies like **Apache, Nginx, and cloud services (AWS, Azure, Google Cloud)**. This choice ensures the game can be accessed across **Windows, Mac, Linux, and mobile devices** without compatibility issues.
2. **Operating Systems Architectures**:

* **Presentation Layer:** The game runs in a **web browser (client-side)** on Windows, Mac, Linux, and mobile.
* **Application Layer:** The **backend logic** runs on Linux cloud servers, processing game requests.
* **Database Layer:** A cloud-based **relational database (MySQL or PostgreSQL)** manages game data, players, and teams.

1. **Storage Management**:

 **Game state storage** (keeping track of game progress).

 **Player profiles & history** (ensuring continuity).

 **Log files and backups** (for debugging and security).

1. **Memory Management**:

 **Garbage Collection:** Automated memory management prevents memory leaks.

 **Load Balancing:** Distributes requests across multiple servers to avoid overload.

 **Caching (Redis, Memcached):** Frequently used game data is stored in **RAM** for fast retrieval, reducing database queries.

 **Virtual Memory Management:** The system dynamically allocates RAM as needed for concurrent users.

1. **Distributed Systems and Networks**:

* **WebSockets (for live communication):** Ensures low-latency updates for gameplay.
* **RESTful APIs:** Connects the frontend game UI to backend services.
* **Content Delivery Networks (CDN):** Ensures fast loading by **distributing content globally**.
* **Auto-Scaling Cloud Servers:** Adjusts resources based on user demand.
* **Failover & Redundancy:** Backup systems prevent downtime in case of server failure.

1. **Security**:

* **Data Encryption (SSL/TLS):** Ensures all communication between players and servers is secure.

**OAuth 2.0 & JWT Authentication:** Secure login system with token-based authentication.

**Role-Based Access Control (RBAC):** Restricts access to sensitive admin functions.

**DDoS Protection (Cloudflare, AWS Shield):** Prevents malicious attacks on the game servers.

**Regular Security Patching:** Automated updates to fix vulnerabilities.

**User Data Protection (GDPR, CCPA Compliance):** Ensures compliance with global data privacy laws.