

The Gaming Room Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.1

## 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 |
| --- | --- | --- | --- |
| 1.1 | 03/17/2025 | Joven Rosales | Implemented Singleton pattern with a private constructor and getInstance() method. |
| 1.1 | 03/17/2025 | Joven Rosales | Used a for-each loop to check if a game with the same name already exists before creating a new one. |
| 1.1 | 03/17/2025 | Joven Rosales | Used a for-each loop to find a game by its id. |
| 1.1 | 03/17/2025 | Joven Rosales | Used a for-each loop to find a game by its name. |
| 1.1 | 03/17/2025 | Joven Rosales | Replaced null with GameService.getInstance() |
| 1.1 | 03/17/2025 | Joven Rosales | At singletonTester.java Replaced null with GameService.getInstance() |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to create a web-based version of their popular game, Draw It or Lose It, which is currently only available as an Android app. This new version will allow players to join from different devices and play online.

## Requirements

*To develop the web-based version of Draw It or Lose It, the following requirements must be met:*

**Functional Requirements**

* Multiplayer Support – The game must allow multiple teams, with each team containing multiple players.
* Unique Names – Each game and team must have a unique name to prevent duplicates.
* Single Game Instance – Only one instance of the game should exist at any given time.
* Unique Identifiers – Each game, team, and player must have a unique identifier to keep track of them properly.
* Turn-Based Gameplay – Teams will take turns guessing images within a set time limit.
* Time Constraints – Each game round will last one minute, with drawings being completed at the 30-second mark. If a team does not guess in time, other teams get a chance to answer within 15 seconds.

**Technical Requirements**

* Web-Based Architecture – The game must be accessible from web browsers on different devices.
* Singleton Pattern Implementation – The GameService class must ensure only one game runs at a time.
* Iterator Pattern Usage – The system should check for unique names before adding new games or teams.
* Entity Class Design – The Game, Team, and Player classes should inherit from a shared Entity class for efficiency.
* Scalability – The software should be designed to handle multiple teams and players efficiently without performance issues.
* Data Persistence – Game data should be stored securely to maintain game progress and user information.

By ensuring these requirements are met, the game will function smoothly and provide an engaging experience for players.

## [Design Constraints](#_2et92p0)

When developing the web-based game, we must consider certain limitations that may affect the design and functionality:

* Platform Compatibility: The game must run smoothly on different web browsers and devices, which means we need to use web-friendly technologies like HTML, CSS, JavaScript, and a backend framework.
* Real-Time Performance: Since the game is time-based, we need a responsive system that updates in real-time without lag. This requires efficient data handling and possibly WebSocket connections.
* Scalability: The game must handle multiple users at once without performance issues, so we need a backend that supports concurrent connections.
* Data Security: Players' information, game progress, and team details must be securely stored and protected from unauthorized access.
* Network Dependency: Since it is an online game, players need an internet connection. We should design the system to handle network interruptions gracefully, ensuring that progress is not lost if a player disconnects.

## [System Architecture View](#_ilbxbyevv6b6)

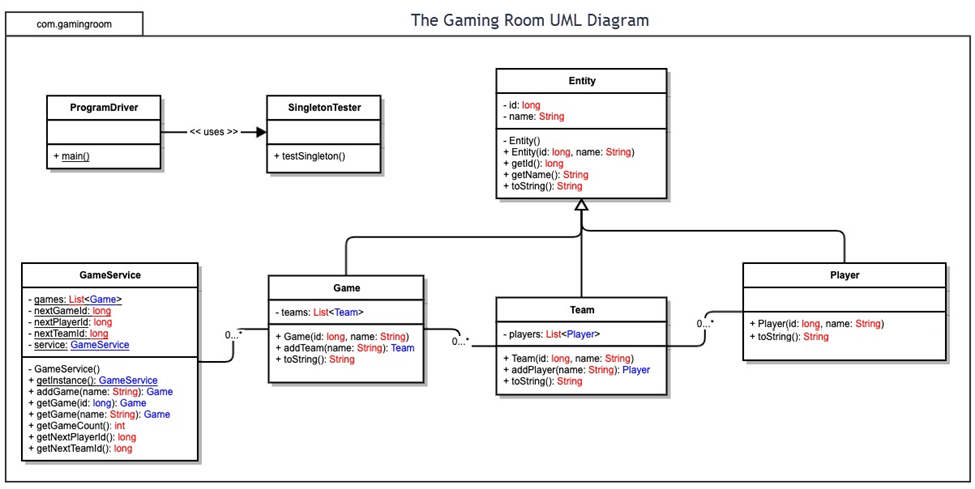
The system will follow a client-server architecture with the following key components:

* Client Side (Frontend): The user interface will be built using HTML, CSS, and JavaScript, ensuring cross-platform compatibility. Players will interact with the game through a web browser.
* Server Side (Backend): A backend service, developed using Java, will handle game logic, user authentication, and real-time updates.
* Database: A database will store game sessions, player details, team names, and scores to ensure data persistence.
* Networking: WebSocket’s or REST APIs will enable communication between the client and server, ensuring smooth real-time interactions.

## 

## [Domain Model](#_8h2ehzxfam4o)

The Domain Model represents the key elements of the game and how they interact. Based on the UML diagram, the main components are:

****

**Evaluation**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are not as commonly used for hosting web applications, but they provide a stable Unix-based environment with good security. However, Mac servers are costly and have limited support compared to Linux. | Linux is the most popular choice for web-based applications due to its stability, security, and open-source nature. It supports a wide range of server software and is highly customizable. However, it requires more technical expertise to set up and manage. | Windows Server offers good support for enterprise applications, especially those built on Microsoft technologies like .NET. It has a user-friendly interface but comes with higher licensing costs and potential security concerns. | Mobile devices are not suitable for hosting web applications due to hardware limitations and lack of server-grade capabilities. However, cloud-based hosting solutions can be accessed through mobile apps. |
| **Client Side** | Developing for Mac requires ensuring compatibility with macOS and Safari. Mac users tend to use high-resolution screens, so UI optimization is necessary. Development tools like Xcode can help, but testing on multiple Mac models is important. | Linux users often rely on open-source browsers like Firefox and Chromium. Web applications need to be optimized for different distributions, and some proprietary software may not work properly. | Windows is the most commonly used client OS, meaning web applications must be tested across multiple versions. Compatibility with Edge, Chrome, and Firefox is important, and UI elements should be optimized for touch and non-touch devices. | Mobile devices require responsive design to adapt to various screen sizes. Applications must be tested on both iOS and Android devices using frameworks like React Native or Flutter for cross-platform compatibility. |
| **Development Tools** | Development for Mac can be done using Xcode, Visual Studio Code, and JetBrains tools. Common languages include Swift, JavaScript, and Java for cross-platform applications. | Linux supports a variety of programming tools such as Eclipse, Visual Studio Code, and JetBrains IDEs. Popular development languages include Python, Java, JavaScript, and PHP. | Windows offers Visual Studio, .NET framework, and a wide range of IDEs. It supports multiple languages, including C#, Java, Python, and JavaScript. | Mobile development is typically done using Android Studio for Android and Xcode for iOS. Cross-platform development can be done using Flutter, React Native, or Unity for game development. |

**Recommendations**

To help Draw It or Lose It work on many different devices like phones, tablets, and computers, the best system to run the game is Linux. Linux is free, safe, fast, and works really well for websites and online apps. A lot of big companies like Google and Facebook use Linux to run their services. For example, the game can be set up on Amazon Web Services (AWS) using Ubuntu Linux, which allows it to grow easily and support more and more players as it becomes popular (Silberschatz et al., 2018).

Linux is designed in a way that keeps all the important parts of the system working together closely, which makes it fast and strong. This is called a monolithic kernel, and it helps everything run smoothly. For instance, when a bunch of people are drawing and guessing at the same time, Linux can quickly keep up with everyone without freezing. It has a built-in tool called epoll, which helps it watch over many connections at once—kind of like a good host keeping track of lots of guests at a party (Tanenbaum & Bos, 2015).

To store the game’s data—like player names, scores, and drawings—it’s best to use a database system like PostgreSQL or MySQL. These systems are like digital filing cabinets that keep everything organized and safe. For example, when a team finishes a game, the scores are saved in the database, and the drawings can be saved in Amazon S3, which stores files online. This way, even if the game restarts or crashes, nothing gets lost (Connolly & Begg, 2015).

Linux is also really smart about how it uses memory, which is the computer’s short-term thinking space. It uses techniques like virtual memory (using part of the hard drive when real memory is full), paging (splitting tasks into smaller chunks), and caching (keeping frequently used info ready to go). These tricks help the system run many tasks at once without slowing down. For example, if ten teams are playing at once, Linux will give each game its own fair amount of memory. Also, by using Docker, we can make sure each game runs separately in its own space so one game won’t mess up the others (Silberschatz et al., 2018).

To let players from all kinds of devices play together, we need the game to work like a distributed system, where different parts of the app talk to each other over the internet. We can use REST APIs to send and receive info (like scores and usernames) and WebSockets for real-time updates (like live drawings). For example, when someone draws a picture, everyone else can see it instantly. To keep everything safe, the game should use HTTPS (which locks and protects data during transfer), OAuth 2.0 for secure logins, and encrypt sensitive data. We should also regularly test the game for security issues so players’ personal information stays safe (Stallings, 2020).

References:

Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). Pearson.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.

Stallings, W. (2020). Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud. Pearson.

Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Pearson.