

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 | 07/13/2025 | Jessica Marchetti | First iteration |
| 1.1 | 08/03/2025 | Jessica Marchetti | Second Iteration |
| 1.2 | 08/14/2025 | Jessica Marchetti | Third Iteration |

**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)

This Software Design Document outlines the proposed architecture and functionality for *Draw It or Lose It*, a web-based multiplayer game being developed for The Gaming Room. The game is modeled after the TV show *Win, Lose, or Draw*, where players guess images being drawn in real time. The application will utilize a predefined library of stock illustrations as prompts, supporting four structured rounds of team-based gameplay. Web-based implementation ensures cross-platform compatibility, allowing access via desktops, tablets, and mobile devices through standard web browsers. The design focuses on meeting client requirements by prioritizing scalability, responsive UI design, real-time interaction, and a user-friendly interface to deliver engaging and reliable gaming experience.

## Requirements

1. *The game must be delivered via a web-based platform, ensuring compatibility across multiple devices and operating systems, including desktops, tablets, and smartphones.*
2. *Each game session must support one or more teams, with functionality to assign multiple players to each team dynamically.*
3. *Unique names must be enforced for both games and teams to avoid naming conflicts. Users should be able to verify name availability during the creation process.*
4. *The application must restrict gameplay to a single active instance in memory at a time. This can be managed using unique identifiers (UUIDs) for games, teams, and players.*
5. *Each game round must include a fixed time limit—such as one minute per round—with drawings gradually revealed up to the 30-second mark to increase challenge and engagement.*
6. *If the primary team fails to solve the puzzle within the time limit, all remaining teams should be given a single opportunity to submit a guess within a 15-second window.*

## [Design Constraints](#_2et92p0)

1. Web**-Based Distributed Platform**:  
   The game application must be designed for deployment on a web-based platform, considering the challenges associated with distributed systems. These include network latency, secure data transmission, and ensuring compatibility across various browsers and device types.
2. **Enforced Name Uniqueness**:  
   To prevent naming conflicts and maintain a seamless user experience, the system must enforce the uniqueness of game, team, and player names during creation and registration processes.
3. Single **Active Game Instance**:  
   The system architecture must ensure that only one active instance of the game service exists in memory at any given time. This constraint is critical to maintaining consistent game state and preventing concurrency issues and can be managed by unique session identifiers and controlled instantiation.

## [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 diagram below shows the main classes that make up the *Draw It or Lose It* game application. At the top is the Entity class, which acts as a base class for all other objects in the system. It includes common properties like id and name, so every object in the game—whether it’s a game, a team, or a player—will have those basic details.

The Game, Team, and Player classes all inherit from Entity, meaning they get those shared features automatically. A Game includes several Teams, and each Team includes multiple Players, creating a clear structure of how the game works.

The GameService class is responsible for managing all the games. It has a strong connection to the Game class, meaning it not only stores games but also controls how they’re created and managed. Similarly, Game manages its teams, and each Team manages its players—so when a game ends, its teams and players go with it.

The ProgramDriver class contains the main method, which is where the program starts running. It creates a single instance of GameService, following the singleton pattern to make sure only one version of that service exists. This class handles adding games, teams, and players through that one instance. It also uses the Singleton Tester class, as shown by the arrow labeled <<uses>> in the diagram.

The class design highlights several object-oriented programming principles:

* **Inheritance**: Game, Team, and Player all inherit from Entity, which helps avoid repeating code and keeps things consistent.
* **Encapsulation**: Classes like GameService keep their data private and only let outside code interact with it through specific methods, helping protect and manage the app’s state.
* **Abstraction**: The system hides complex details about how games, teams, and players are managed and focuses instead on the key tasks each class needs to handle.

Overall, this structure helps keep the code organized, easier to maintain, and more scalable as the game grows.

**"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** | Unix platforms provide a stable and secure operating system environment well-suited for hosting web applications. They feature mature system architectures and robust support for enterprise-level security protocols. The ecosystem includes comprehensive development tools and APIs that developers rely on for performance and reliability. However, Unix systems generally demand higher hardware costs due to proprietary requirements and offer more limited scalability compared to Linux and Windows, especially in cloud-native and distributed computing contexts. | An open-source OS that provides extensive customization and flexibility, supported by a vast ecosystem of software and development tools. It is highly scalable and renowned for its stability, security, and suitability for a wide range of deployment environments, from embedded systems to large-scale servers. However, it may present limitations in graphical user interface (GUI) features and occasionally faces challenges with hardware compatibility, particularly with proprietary or niche devices. | Offers extensive software compatibility and benefits from a strong developer community. Provides broad hardware support along with comprehensive documentation and technical resources. However, it is associated with a higher exposure to known security vulnerabilities compared to some other platforms. | Designed for portability with touchscreen and gesture-based input, operating within the constraints of limited screen size. Hardware capabilities can vary significantly across devices. |
| **Client Side** | An intuitive and user-friendly interface minimizes the learning curve for users. However, developing and maintaining multiple client applications can increase development time and costs, and may necessitate a broader range of technical expertise. | Although the software is open-source and free to use and distribute, associated costs such as hardware acquisition and specialized development tools must be considered. The platform often presents a steep learning curve due to its complexity and flexibility. Additionally, supporting multiple client environments necessitates expertise across various technology stacks and development frameworks. | Licensing costs for proprietary software are generally higher than those for open-source options, as they often include fees for commercial support, maintenance, and usage rights. | Critical factors to consider include the implementation of responsive design frameworks to optimize user experience across varying device form factors and network conditions. Furthermore, integration with native device APIs—such as camera access, GPS location services, and push notification systems—is essential to fully utilize platform-specific capabilities and improve application functionality. |
| **Development Tools** | Node.js and JavaScript are widely adopted technologies for development. Common integrated development environments (IDEs) include Visual Studio Code (VSCode) and Xcode. | The platform boasts a rich ecosystem featuring popular IDEs such as Visual Studio Code (VSCode), Atom, and Sublime Text. It also provides a robust command-line interface complemented by advanced package management systems like apt and yum for efficient software installation and maintenance. | C# and the .NET framework are widely used for developing Windows-based web applications. Popular integrated development environments (IDEs) for this ecosystem include Visual Studio and JetBrains Rider. | Java and JavaScript, Kotlin, Swift, and Objective-C are commonly used programming languages for mobile application development. Development environments typically include Android Studio for Android and Xcode for iOS, complemented by device emulators and simulators for testing and debugging across various hardware configurations. |

## 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**: To enable The Gaming Room to extend *Draw It or Lose It* across multiple computing environments, adopting a web-based platform is recommended. This approach allows the game to be accessed and played on a wide range of devices equipped with web browsers, including desktops, laptops, tablets, and smartphones. By leveraging modern web technologies, the game can reach a broader audience while delivering consistent user experience across diverse platforms.
2. **Operating Systems Architectures**: For the selected web-based platform, the system architecture will primarily utilize client-server and web technologies. The client-side will be built using HTML, CSS, and JavaScript to render the game interface and manage user interactions. On the server-side, multi-tier architecture will be implemented, comprising presentation, application, and data layers to ensure modularity, scalability, and maintainability.
3. **Storage Management**: An optimal storage solution would combine a relational database management system (RDBMS) with cloud-based storage services. The RDBMS would manage structured data, including game progress, user profiles, and game statistics, providing efficient querying and transactional support. Meanwhile, cloud storage would be utilized for media assets, such as stock images, offering scalable, durable, and platform-independent access to these resources.
4. **Memory Management**: The proposed web-based platform leverages automatic memory management capabilities inherent in modern web browsers. These browsers utilize garbage collection algorithms to manage memory allocation and deallocation dynamically, relieving developers from the complexities of manual memory management. This approach promotes efficient memory usage and helps prevent memory leaks, thereby enhancing application stability and performance.
5. **Distributed Systems and Networks:** To facilitate cross-platform communication, the game will adopt a distributed software architecture supported by network connectivity. This involves deploying a centralized server or cloud-based infrastructure that acts as a communication hub for all game clients. The server will manage game state synchronization, real-time updates, and message exchanges among players across diverse devices. Additionally, the system must incorporate robust error handling and synchronization mechanisms to mitigate the impact of network issues such as intermittent connectivity and limited bandwidth, ensuring consistent gameplay experiences.
6. **Security**: To safeguard user information across multiple platforms, several security measures should be implemented. The recommended web-based platform must support secure communication protocols, such as HTTPS/TLS, to protect data in transit. User authentication and authorization mechanisms—like username and password systems or multi-factor authentication—should be used to control access to game features and user profiles. Additionally, data encryption should be applied both for sensitive information stored in databases and for data transmitted over the network to ensure confidentiality and integrity.