

Draw It or Lose It (Web-Based Version)

# **CS 230 Project Software Design Template**

Version 1.2

Jason Perez

June 22, 2025

## 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 | 05/27/2025 | Jason Perez | Initial draft of the software design document |
| 1.1 | 06/08/2025 | Jason Perez | Evaluation section revised/expanded |
| 1.2 | 06/22/2025 | Jason Perez | Recommendations section revised/expanded |

**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 expanding their Android-exclusive game, Draw It or Lose It, by developing a web-based version that can operate across various platforms. The game offers a collaborative drawing and guessing experience, where each round is time-limited, and teams must make guesses based on the images created.

To promote efficiency, scalability and centralized management, the software will implement the Singleton pattern for overseeing the game engine service and the Iterator pattern to maintain entity uniqueness. The backend is being developed in Java using an object-oriented framework, providing flexibility and future features and integrations. This document presents the design strategy aimed at creating a version of the game that is both maintainable and scalable, aligning with business requirements.

## Requirements

Business Requirements include the following:

* Enable web access for the game across multiple platforms
* Ensure a single game instance is running at any time
* Support multiple teams and players per game
* Enforce unique names for games, teams, and players

Technical Requirements include the following:

* Use object-oriented programming principles
* Implement Singleton and Iterator patterns
* Provide backend support for entity management (game, team, player)
* Allow the environment to scale and support distributed systems

## [Design Constraints](#_2et92p0)

* Web-based distributed environment: The application must handle simultaneous access across different platforms
* Singleton Pattern: Only one instance of the GameService should exist to manage game state consistently
* Unique names: Games, teams, and players must have unique names to avoid collisions during gameplay
* Scalability: Architecture must accommodate future enhancements like user authentication, real-time synchronization, or multiplayer chat
* Platform independence: Codebase should support deployment on various OS and server platforms, requiring abstraction from OS-specific dependencies

## [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 for the application includes the following classes:

* Entity: A base class for Game, Team, and Player, containing shared attributes like id and name
* Game: Manages a list of teams
* Team: Manages a list of players
* Player: Represents an individual user
* GameService: Singleton class managing the creation and retrieval of games

OOP Principles:

* Inheritance: Game, Team, Player inherit from Entity
* Encapsulation: Data is hidden using private variables and exposed via public getters
* Abstraction: High-level methods like addGame() and addTeam() hide internal list management
* Design Patterns: Singleton ensures a single shared instance of the game engine; Iterator is used to check for existing entities before creation

**"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 is UNIX-based and is capable of hosting Java-based web applications by utilizing tools such as Apache or Nginx; however, it is not typically favored for large-scale server deployments. Furthermore, Apple has ceased official support for macOS Server, and the hardware requirements contribute to higher hosting expenses. | Linux serves as the industry benchmark for server environments. It is exceptionally scalable, reliable, and is free of licensing fees. Additionally, it accommodates a range of web servers (such as Apache and Nginx), containerization platforms (like Docker), and orchestration tools (including Kubernetes), rendering it perfect for extensive distributed applications. | Windows Server is capable of running Java and hosting web applications through IIS or alternative third-party servers. Nevertheless, it necessitates licensing fees and frequently results in elevated maintenance expenses. It is less frequently utilized for high-scale web hosting. | Mobile devices are not appropriate for server hosting because of their restricted CPU and memory resources. Their function is exclusively client-side, utilizing responsive browser interfaces to access the application. |
| **Client Side** | macOS users utilize the application via Safari, Chrome, or Firefox. The browser-based delivery guarantees compatibility; however, testing must consider the unique quirks of Safari and the scaling on Retina screens. It is crucial to implement a responsive user interface using frameworks such as Bootstrap or Tailwind CSS. | Linux clients will engage with the game through browser-based access (such as Chrome, Firefox, etc.). Testing must verify compatibility across different distributions and various window managers. Web applications should refrain from incorporating OS-specific dependencies. | Windows is the most commonly utilized desktop operating system, therefore ensuring compatibility with Chrome, Edge, and Firefox is essential. A responsive design will facilitate seamless performance across different screen resolutions. | The application must function effectively on both iOS Safari and Android Chrome. The user interface should be designed with a mobile-first approach, ensuring it is responsive and lightweight to facilitate smooth performance, even on lower-end devices. Particular emphasis should be placed on screen real estate, touch interactions, and the differing behaviors of various browsers. |
| **Development Tools** | Common tools consist of IntelliJ IDEA, Eclipse, and VS Code. Xcode is essential for native iOS development; however, it is not necessary for this web-based application. There are no licensing cost associated with IDEs unless premium editions are utilized. Testing on Apple hardware might be required. | Linux offers comprehensive support for a variety of open-source Integrated Development Environments(IDEs) such as Eclipse, IntelliJ, and VS Code. It is highly effective for DevOps tools, scripting, continuous integration and continuous deployment (CI/CD) pipelines, as well as container management. This makes it a robust option for both software development and server deployment. | Windows is compatible with all leading Java Integrated Development Environments (IDEs). Both Visual Studio Code and IntelliJ function effectively; however, their significant consumption of system resources may pose a disadvantage. Certain enterprise-grade tools might necessitate a license. Further testing is required to address browser inconsistencies in Edge. | Android Studio serves as the primary tool for native Android development; however, the necessity for this is diminished with web-based delivery unless a native wrapper is taken into account. In the case of iOS, Xcode is essential for the construction or testing of native applications. To ensure browser compatibility, front-end developers may utilized cross-platform frameworks such as React and Flutter (with web support), and they can conduct testing using platforms like BrowserStack. |

## 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**: In the server environment that supports Draw It or Lose It, Linux, particularly Ubuntu Server, is the most suitable operating system. It is open-source, highly scalable, cost-efficient, and recognized for its security and dependability. Furthermore, Linux provides extensive compatibility with Java and contemporary web development frameworks. Moreover, Linux integrates effortlessly with cloud platforms like AWS, Google Cloud Platform (GCP), and Microsoft Azure, facilitating efficient deployment and future scalability.
2. **Operating Systems Architectures**: Linux utilizes a monolithic kernel architecture, facilitating high-performance communication between the kernel and user space. It accommodates multithreading and process isolation, which are crucial for gaming applications that demand responsiveness and concurrency. Additionally, the architecture is highly customizable, allowing for the implementation of container-based microservices for backend components. Technologies such as Docker and Kubernetes can be utilized to isolate individual game services, scale them independently, and guarantee efficient resource management across various platforms.
3. **Storage Management**: For the reliable and persistent storage of game data, including game sessions, teams and player information, it is advisable to utilize a relational database such as PostgreSQL. This recommendation is based on its proven reliability, robust support for data integrity, and adherence to ACID principles. Alternatively, if scalability and schema flexibility are of utmost importance, MongoDB can be employed for its adaptable document-based storage capabilities. In the case of static resources, such as user-generated images and avatars, it is recommended to utilize cloud object storage solutions like Amazon S3. This approach guarantees durability, high availability, and global accessibility, thereby ensuring consistent user experience across various devices and regions.
4. **Memory Management**: The Linux operating system, when paired with the Java Virtual Machine (JVM), offers strong memory management capabilities through methods like garbage collection, heap allocation, and thread-local storage. Optimizing the JVM can enhance memory utilization for Draw It or Lose It, particularly in handling multiple player sessions concurrently. Implementing the Singleton design pattern for the GameService class guarantees reduced memory usage by preserving a single instance of the game controller for the duration of the application. Moreover, Java’s integrated memory profiling tools and effective resource deallocation can help prevent memory leaks.
5. **Distributed Systems and Networks**: To facilitate communication across different platforms, a distributed systems methodology will be implemented. RESTful APIs will manage the majority of client-server interactions, whereas WebSockets will provide real-time updates for both gameplay and chat features. These services may be deployed utilizing microservices architecture, allowing each functional component (authentication, game logic, user management) to function independently. Load balancers will effectively distribute traffic among servers, and fault-tolerant systems will guarantee minimal downtime during outages. Caching solutions like Redis can be employed to enhance latency and support scalable user interactions in real-time.
6. **Security**: User protection is of utmost importance and will be implemented through a layered security model. All data transmitted between clients and servers will be secured using HTTPS and TLS encryption. Authentication will be managed via OAuth 2.0, while session integrity and access control will be upheld through the use of JSON Web Tokens (JWTs). On the server side, Linux provides strong firewall tools (such as iptables and ufw) and security frameworks like AppArmor or SELinux, which can enforce security policies at the operating system level. Data at rest should be protected through file system-level encryption or database encryption extensions. Furthermore, all user input must undergo validation and sanitization to mitigate common web vulnerabilities, including SQL injection and cross-site scripting (XSS). Regular security audits and penetration testing should be conducted to maintain the ongoing integrity of the platform.