

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 | 1/23/2025 | Jarrod Schantz | Initial draft completed with all required sections outlined and addressed. |

**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 seeks to expand its popular Android game, "Draw It or Lose It," into a web-based platform accessible across multiple devices. Currently, the game is limited to a single-platform application. This project aims to provide a scalable, efficient, and secure solution that allows the game to operate seamlessly across diverse computing environments while meeting the unique requirements of the client. To address these challenges, this software design document outlines a plan leveraging object-oriented programming principles and established software design patterns, including Singleton and Iterator. These patterns ensure efficient management of game instances, unique identification of teams and players, and streamlined development processes. The proposed design emphasizes scalability, maintainability, and user experience, laying a solid foundation for future enhancements and expansion.

## Requirements

* Enable the creation and management of one or more teams, each with multiple players.
* Ensure all game, team, and player names are unique, allowing users to check name availability.
* Maintain a single active instance of the game in memory at any given time.
* Develop a web-based application capable of supporting multiple platforms with high performance and reliability.
* Provide a scalable and secure solution for distributed gameplay.

## [Design Constraints](#_2et92p0)

1. **Platform Compatibility:** The application must support multiple operating systems (Windows, Mac, Linux) and mobile devices, requiring careful consideration of cross-platform frameworks and technologies.
2. **Unique Identification:** Unique identifiers for games, teams, and players must be generated and managed efficiently to ensure data consistency across sessions.
3. **Real-Time Performance:** The game’s real-time nature necessitates minimal latency for rendering images and handling user interactions, impacting both server and client-side design.
4. **Security Requirements:** User data and gameplay must be protected against breaches, requiring robust encryption, authentication, and secure communication protocols.
5. **Scalability:** The system must handle increasing numbers of users and games concurrently, necessitating scalable architecture and efficient resource management.

## [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 for the application leverages inheritance to promote code reusability and maintainability. The base Entity class encapsulates common attributes such as id and name, which are inherited by the Game, Team, and Player classes.

* **Game Class:** Represents the game instance, managing the state of gameplay and holding a list of participating teams.
* **Team Class:** Encapsulates a group of players and their associated attributes.
* **Player Class:** Represents individual users, inheriting the unique identification mechanisms from the Entity class.

Object-oriented principles such as encapsulation ensure that attributes are accessible only through defined methods, enhancing data integrity. The Singleton pattern ensures only one instance of GameService exists, centralizing game management. The Iterator pattern is employed to traverse collections of games, teams, and players efficiently, ensuring unique naming and smooth operations.

**"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 offers a reliable but costly option for hosting web-based applications. It supports a variety of tools but may require additional expertise for server optimization. | Linux provides robust, cost-effective hosting with excellent support for scalability and customization, making it ideal for web applications. | Windows servers are user-friendly and widely supported, though licensing costs are higher compared to Linux. | Mobile devices lack robust server capabilities but are integral for client-side interaction and accessibility. |
| **Client Side** | Development for macOS clients requires specialized expertise and tools like Xcode, increasing time and cost. | Linux clients often require less development time due to lightweight environments and open-source tools. | Windows clients are straightforward to develop using Visual Studio and similar tools, balancing cost and time efficiency. | Mobile client development involves using frameworks like React Native, balancing cost and cross-platform compatibility. |
| **Development Tools** | Languages such as Swift and frameworks like Xcode are necessary for macOS deployment. | Linux relies on programming languages such as Python, Java, and C++ with tools like Eclipse or IntelliJ. | Windows development often involves C#, .NET, or Java, using IDEs like Visual Studio. | Mobile development employs frameworks like Flutter, React Native, or Swift for iOS, with IDEs like Android Studio. |

## 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:**

For server-side hosting, **Linux** is the recommended operating system. Linux offers cost-effective scalability, robust security, and a wealth of open-source tools that support distributed applications. For client-side development, using cross-platform frameworks such as **React Native** or **Flutter** enables the game to run on mobile devices (iOS, Android) as well as desktop platforms (Windows, macOS, Linux). This dual approach ensures broad accessibility while minimizing development overhead.

1. **Operating Systems Architectures:**

Linux employs a modular, monolithic kernel that supports high-performance multitasking and efficient resource management. This makes it particularly well suited for hosting web-based backends that require rapid handling of concurrent user requests. In contrast, client-side architectures must address the specific requirements of each platform—desktop operating systems typically leverage their native GUI toolkits, while mobile devices rely on frameworks that optimize touch interfaces and battery life. By designing the game’s backend with Linux and the front-end with cross-platform frameworks, you create a cohesive system architecture that bridges diverse environments.

1. **Storage Management:**

The design document recommends utilizing cloud-based storage solutions such as **AWS S3** or **Google Cloud Storage**. These solutions provide scalable, redundant, and highly available data storage. In addition, leveraging local file systems like ext4 or XFS on the Linux server can be combined with distributed file systems (such as **Ceph** or **GlusterFS**) for managing game data. This hybrid approach ensures that large collections of game assets, user profiles, and real-time game states are accessible and resilient across distributed systems.

1. **Memory Management:**

Linux’s advanced memory management system, which includes virtual memory, paging, and caching mechanisms, ensures optimal resource utilization even under heavy load. For Draw It or Lose It, these techniques are critical in maintaining smooth performance during gameplay. Additionally, when using a Java-based framework like Dropwizard (as seen in the provided project code), the Java Virtual Machine (JVM) adds its own garbage collection and memory management processes. This layered approach—combining Linux’s native memory techniques with JVM optimizations—ensures that the game operates reliably without memory leaks or performance degradation.

1. **Distributed Systems and Networks:**

To facilitate communication between various platforms, the game should be architected as a distributed system. The backend can expose **RESTful APIs** (and potentially use WebSocket protocols for real-time features) that enable seamless interactions between the game server and clients. Incorporating load balancers, redundancy mechanisms, and service discovery protocols will ensure that even if parts of the network experience outages, overall system performance remains resilient. These distributed networking techniques not only support scalability but also enhance the user experience through reduced latency and improved fault tolerance.

1. **Security:**

* Security is a paramount concern for user data protection and overall system integrity. The following measures are recommended:
* **Authentication and Authorization:** Implement robust security frameworks such as HTTPS, OAuth, or token-based authentication. The provided design leverages Dropwizard’s Basic Auth, and role-based access control (RBAC) should be enforced to distinguish between regular users and administrators.
* **Encryption:** Ensure that all data transmitted between clients and servers is encrypted using TLS/SSL protocols. Data stored on disk (whether locally or in the cloud) should be encrypted using industry-standard algorithms.
* **Platform-Specific Security Enhancements:** Linux offers built-in security modules like SELinux or AppArmor, which provide additional layers of access control. Regular security audits, timely patching, and adherence to best practices (e.g., least privilege access) are essential.
* **Network Security:** Use firewalls, intrusion detection systems, and network segmentation to protect against external threats. This comprehensive strategy minimizes the risk of data breaches and secures the overall communication infrastructure.