

The Gaming Room

# **CS 230 Project Software Design Template**

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 | 02/9/25 | Josh | 2nd edit update |

**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 their Android-based game, "Draw It or Lose It," to a web-based, multi-platform environment. This requires careful planning and implementation of software design patterns to ensure efficiency, scalability, and usability. Key challenges include maintaining unique identifiers for games, teams, and players, as well as ensuring that only one instance of the game service operates in memory. By using proven design patterns like Singleton and Iterator, we can ensure robust and maintainable software architecture. This document outlines the software design plan, domain model, and implementation strategy for the project.

## Requirements

*The client requires a web-based version of the game that supports:  
- Unique identifiers for games, teams, and players.  
- Only one instance of the game service in memory.  
- Unique game and team names to prevent duplicates.  
- Multiple teams and players within a game.  
- Cross-platform compatibility to reach broader audiences.*

## [Design Constraints](#_2et92p0)

Developing the game application for a web-based distributed environment presents several design constraints:  
  
- \*\*Unique Identifiers\*\*: Each game, team, and player must have a unique name or identifier to avoid conflicts.  
- \*\*Single Instance\*\*: Only one instance of the `GameService` class can exist at any time, ensuring centralized management of game data.  
- \*\*Cross-Platform Compatibility\*\*: The game must run seamlessly on various platforms, including web, mobile, and desktop environments.  
- \*\*Performance and Synchronization\*\*: Distributed environments require real-time updates and synchronization, which may introduce latency.  
  
These constraints necessitate efficient design patterns like Singleton for instance control and Iterator for managing unique names.

## [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 shows that the Entity class is the base for Game, Team, and Player, sharing common attributes like id and name. The GameService class is a singleton that manages all games and ensures unique identifiers. Game contains teams, and Team contains players, forming a hierarchy. This design uses inheritance to avoid redundancy and the singleton pattern to centralize management, making the system scalable and efficient.**

**"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 has limited support for server environments. While it includes Apache and PHP, enterprise-grade hosting typically favors Linux or Windows. Licensing costs can be high, as Mac servers are uncommon. | Linux is the dominant choice for web hosting due to its open-source nature, strong security, and flexibility. It supports various web servers like Apache and Nginx. No licensing costs make it highly cost-effective. | Windows Server supports IIS for hosting web applications. It provides enterprise support but comes with significant licensing costs. | Mobile devices do not function as servers but interact with the back-end server to receive data. Hosting is typically handled by cloud-based or traditional server environments. |
| **Client Side** | Web applications on macOS must be optimized for Safari, along with other browsers like Chrome and Firefox. Development costs may increase if targeting native macOS applications. | Linux users primarily use web applications via browsers. Ensuring compatibility with Chrome, Firefox, and other browsers is crucial. Native desktop apps for Linux are less common. | Windows users access web applications through multiple browsers like Edge, Chrome, and Firefox. If native desktop applications are required, development resources will increase. | Mobile applications need to be developed for both iOS and Android. A responsive web design ensures accessibility via mobile browsers, while native apps require additional development efforts. |
| **Development Tools** | Xcode is the primary IDE for macOS development. Cross-platform development can use frameworks like Electron or React Native. Licensing costs for Apple development programs apply. | Linux development typically relies on open-source tools like VS Code, Eclipse, or JetBrains. No licensing fees make it cost-effective. | Windows development often uses Visual Studio, which has both free and paid versions. Windows-specific applications require additional resources. | Mobile app development requires tools like Android Studio for Android and Xcode for iOS. Cross-platform frameworks like Flutter or React Native reduce redundant work. |

## 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 recommended operating platform is Google Cloud Platform (GCP). This platform is ideal for expanding Draw It or Lose It to other computing environments because it supports cross-platform development and deployment, including web, mobile (iOS and Android), and desktop operating systems. GCP provides auto-scaling and load balancing to handle high traffic, ensuring optimal performance. It supports multiple operating systems such as Windows, macOS, Linux, iOS, and Android, allowing seamless integration and deployment. GCP offers integrated development tools like Google App Engine and Firebase, which ease the development and deployment process. Its pay-as-you-go pricing model provides cost control as the user base grows. Additionally, GCP ensures comprehensive security with data encryption, identity management, and DDoS protection, making it a reliable and secure choice for the game’s expansion.
2. **Operating Systems Architectures**: The recommended platform utilizes microservices architecture with containerization. This architecture allows modular development, enabling each component of the game, such as authentication, game logic, leaderboard, and chat, to be developed, deployed, and scaled independently. Docker containers ensure consistent deployment across different environments, maintaining seamless integration and portability. Kubernetes manages container orchestration, providing high availability, load balancing, and efficient resource utilization. This architecture supports deployment on Linux-based servers for cost efficiency and stability while maintaining compatibility with Windows and macOS for development and testing.
3. **Storage Management**: Google Cloud Storage combined with Firestore is recommended for dynamic data storage. Cloud Storage is used for object storage of game assets, such as images and audio files, ensuring fast loading times. Firestore, a NoSQL database, efficiently stores dynamic content like game states and user progress, allowing real-time updates and scalability. The system includes built-in replication and backup features that provide data integrity and disaster recovery. In-memory caching is handled by Cloud Memorystore (Redis), which caches frequently accessed data such as leaderboards, enhancing performance and reducing latency.>
4. **Memory Management**: The recommended operating platform employs dynamic memory allocation and advanced caching mechanisms to efficiently manage memory. Dynamic memory allocation allows the system to allocate and deallocate memory as needed, optimizing performance and preventing memory leaks. Automatic garbage collection is implemented to free unused memory resources. The platform uses virtual memory management to handle large game states without exhausting physical RAM. In-memory caching with Redis is utilized to store frequently accessed data, reducing database load and improving response times.
5. **Distributed Systems and Networks**: The system uses a distributed architecture with RESTful APIs and WebSocket connections to enable communication between various platforms. REST APIs handle asynchronous requests and communication between services, while WebSocket enables real-time multiplayer interactions. The platform uses Cloud Load Balancer for efficient traffic distribution and auto-scaling to maintain consistent performance during high traffic. Monitoring tools are integrated for real-time tracking of network health, and failover strategies are implemented to ensure connectivity and service availability during outages. This architecture supports seamless communication between web, iOS, and Android platforms.
6. **Security**: A multi-layered security approach is recommended to protect user information on and between various platforms. Data encryption is enforced using TLS/SSL for secure data transmission and AES-256 for data at rest. User authentication and authorization are managed using OAuth 2.0 and Firebase Authentication, with JSON Web Tokens (JWT) ensuring secure session management. Role-Based Access Control (RBAC) restricts data access based on user roles. Network security is maintained through firewalls, DDoS protection using Google Cloud Armor, and Identity and Access Management (IAM) for granular permission controls. The platform complies with data protection regulations, including GDPR and CCPA, ensuring user data privacy and security.