

The Gaming Room

# CS 230 Project Software Design Template

Version 1.3

## Table of Contents

[CS 230 Project Software Design Template 1](#_Toc115077317)

Table of Contents 2

[Document Revision History 2](#_Toc115077319)

[Executive Summary 3](#_Toc115077320)

[Requirements 3](#_Toc115077321)

[Design Constraints 3](#_Toc115077322)

[System Architecture View 3](#_Toc115077323)

[Domain Model 4](#_Toc115077324)

[Evaluation 4](#_Toc115077325)

[Recommendations 6](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.3 | 2/23/2025 | Steven Gifford | Final draft of software design document |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is seeking to expand their existing mobile game, *Draw It or Lose It*, into a web-based platform that can serve multiple operating systems. The game requires multiple teams to participate, team and player names must be unique to avoid duplication. Additionally, only a single instance of the game should exist at a time to ensure consistency. The solution will leverage object-oriented programming principles, particularly the Singleton and Iterator design patterns, to enforce these constraints efficiently. This document outlines the key requirements, design constraints, and technical considerations necessary for developing the game application in a distributed environment.

## Requirements

*The client has the following business and technical requirements:*

* *The game must support multiple teams, with multiple players per team.*
* *Each team and player must have a unique name.*
* *Only one instance of the game should exist in memory at any given time.*
* *The application must be web-based and function on various operating systems.*
* *The game should maintain session persistence for active users.*

## [Design Constraints](#_2et92p0)

1. Singleton Pattern for Game Service
   * The application must ensure that only one instance of the *GameService* class exists at a time to enforce consistency.
   * Implication: This will be implemented using the Singleton design pattern to prevent multiple instances from being created.
2. Iterator Pattern for Searching and Retrieval
   * The system must efficiently search through existing games, teams, and players to enforce unique names.
   * Implication: Using the Iterator pattern will allow traversal through collections to check for duplicate entries before adding new objects.
3. Web-Based Architecture
   * The application must function as a distributed system accessible via a web browser.
   * Implication: The backend must support RESTful APIs for seamless communication between the client and server.

## [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 game application consists of:

* Entity (Base Class): This abstract class holds common attributes (id, name) and behaviors for all game objects.
* Game (Inherits Entity): Contains a list of teams and manages their interactions.
* Team (Inherits Entity): Contains a list of players and manages their interactions.
* Player (Inherits Entity): Represents individual players.
* GameService (Singleton Class): Manages all game instances and ensures only one instance exists at a time.

OOP Principles Used:

* Encapsulation: The data fields in each class are private, with public getter methods ensuring controlled access.
* Inheritance: The Entity base class is extended by Game, Team, and Player, reducing code redundancy.
* Polymorphism: Methods are overridden to allow different behaviors for different entities.

"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 | Mac servers support web hosting through Apache and Nginx but may have compatibility issues with some enterprise applications. The macOS server market share is relatively small, making it a less common choice for large-scale deployments. | Linux is widely used for web servers, supporting Apache, Nginx, and cloud deployments. It is open-source, highly customizable, and generally offers the best performance, security, and stability for web-based applications. | Windows supports IIS (Internet Information Services) and can run Apache and Nginx, but it is often associated with higher licensing costs. While it integrates well with Microsoft tools, it may require additional configuration for optimal performance. | Mobile platforms are generally not used for server-side hosting. Instead, they rely on backend cloud services or dedicated web servers for data processing and API communication. |
| Client Side | macOS users typically access applications through Safari or third-party browsers like Chrome and Firefox. Web-based applications need to be optimized for Safari’s rendering engine (WebKit). | Linux supports all major browsers, including Chrome and Firefox. However, ensuring browser compatibility across different Linux distributions requires testing. | Windows is the most widely used OS for web browsing and supports all major browsers, making it the most accessible platform for users. | Mobile devices require a responsive web design to ensure smooth user experience on smaller screens. The application must be optimized for touch input and performance on both Android and iOS. |
| Development Tools | Xcode is required for iOS development, while IntelliJ IDEA, Eclipse, and Visual Studio Code can be used for web development. Licensing costs may apply for proprietary tools. | Linux supports free and open-source development tools like Eclipse, IntelliJ IDEA, and VS Code. It is preferred for backend development due to its compatibility with server environments. | Windows supports Visual Studio, Eclipse, and IntelliJ IDEA. It is the most common platform for enterprise development but may require additional licensing for some tools. | Android development requires Android Studio, while iOS development requires Xcode. Cross-platform frameworks like React Native and Flutter can help streamline development. |

Recommendations

1. Operating Platform

Linux is the ideal choice for hosting the game application due to its stability, security, and cost-effectiveness. It provides high scalability, making it well-suited for handling thousands of concurrent players. Additionally, Linux’s strong compatibility with major web servers (Apache, Nginx) and cloud platforms ensures flexibility for future expansion. Its open-source nature eliminates licensing costs, making it a budget-friendly solution.

2. Operating System Architectures

A cloud-based, containerized deployment using Docker and Kubernetes is recommended to ensure cross-platform compatibility and scalability. By utilizing microservices architecture, the system can efficiently manage game sessions, player data, and real-time interactions. This modular approach enhances system resilience, allowing for easier updates and maintenance without disrupting active game sessions.

3. Storage Management

A cloud-based relational database such as MySQL or PostgreSQL will be used to store game data securely. This setup ensures high availability and scalability, accommodating future growth. Data redundancy and automated backups will be implemented to prevent data loss and ensure disaster recovery. Additionally, content delivery networks (CDNs) will optimize asset distribution, reducing load times for players.

4. Memory Management

To enhance performance, the application will implement caching strategies using Redis. This will minimize database load and improve response times for frequently accessed game data and player sessions. Lazy loading and memory pooling techniques will be used to optimize resource allocation, ensuring smooth performance even under high user demand.

5. Distributed Systems and Networks

The game will use WebSockets for real-time communication, providing low-latency interactions between players and teams. To enhance responsiveness and optimize game performance across different geographical locations, the system will integrate a content delivery network (CDN). This will reduce server strain and ensure fast, efficient asset loading for players, regardless of their location. Load balancing will further distribute network traffic, preventing bottlenecks and maintaining a seamless user experience.

6. Security

To protect user information across multiple platforms, the application will implement OAuth 2.0 for secure authentication and role-based access control (RBAC) to ensure users can only access authorized game features. All sensitive data, including login credentials and game activity, will be encrypted using SSL/TLS to prevent unauthorized interception during transmission.

To mitigate security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF), the system will incorporate input validation, secure API communication, and intrusion detection systems (IDS). Regular security audits, penetration testing, and vulnerability assessments will help identify and address potential risks proactively. Additionally, multi-factor authentication (MFA) will be implemented to strengthen account security, reducing the likelihood of unauthorized access.

By integrating these security best practices, The Gaming Room will create a safe and reliable gaming environment, ensuring data integrity, user privacy, and protection from cyber threats. This comprehensive security approach will support the game’s expansion across multiple platforms while maintaining a secure and enjoyable user experience.