

**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 | 10/24/24 | Joseph Toj | Refactored Game, Team, and Player classes to inherit from a new Entity base class, implemented the Singleton Pattern in GameService to ensure only one instance exists, and used the Iterator Pattern to manage unique game and team names. |

**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, a new client of Creative Technology Solutions (CTS), is expanding their Android-based game *Draw It or Lose It* into a web-based, multi-platform game. The objective of the initiative is to produce a game application that allows for player and team management, guarantees that there is only one instance of the game at a time, and supports unique game and team names. These needs will be met by this software design, which will make use of the Singleton and Iterator Patterns to guarantee scalability, performance, and data consistency across several platforms

## Requirements

*Business* ***Requirements****:*

* *Support multi-platform game access.*
* *Allow multiple teams, each with multiple players.*
* *Enable unique naming for games and teams.*

*Technical* ***Requirements****:*

* *Implement Singleton Pattern for a single GameService instance.*
* *Ensure unique identifiers for each game, team, and player.*
* *Utilize Iterator Pattern for searching and managing game and team lists.*

## [Design Constraints](#_2et92p0)

**Web-Based Distributed Environment**:

* Ensure compatibility across devices (desktops, tablets, mobile).
* Account for variable network speeds and latency.
* Leverage Singleton Pattern for centralized game state management

**Implications**:

Limited hardware resources, so memory and processing efficiency are critical.

The need for high network reliability to support distributed play.

## [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 illustrates a foundational Entity class with id and name attributes, inherited by Game, Team, and Player classes. This setup promotes **inheritance** and **encapsulation**. The GameService class uses the **Singleton Pattern** to limit its instantiation to one, ensuring centralized game state. The **Iterator Pattern** is implemented in GameService to allow efficient traversal and checking of game and team names, supporting unique identification as required by the client.

**"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** | |  | | --- | | Reliable for web applications but less commonly used for high-scale server deployments. Licensing costs are generally higher than Linux but manageable for smaller-scale applications. Mac servers are known for stability and ease of maintenance, making it suitable for hosting if scalability isn’t a primary concern. |  |  | | --- | |  | | |  | | --- | | Highly reliable, open-source, and widely used for large-scale web applications. Linux is preferred for scalability and cost-effectiveness, as it has no licensing fees and is compatible with cloud-based servers. Linux’s flexibility allows for robust server configuration. |  |  | | --- | |  | | |  | | --- | | Windows servers are widely compatible and user-friendly, but licensing fees can be higher than Linux. It is often preferred in environments that use Microsoft services. Scalability is achievable, but licensing costs may increase with more complex configurations. |  |  | | --- | |  | | |  | | --- | | Supported for lightweight applications but typically requires a cloud-based server backend to manage data and ensure performance. It’s essential to optimize for responsive design and performance due to limited processing resources on mobile devices. |  |  | | --- | |  | |
| **Client Side** | |  | | --- | | Excellent support for web applications on Safari and Chrome. Development on Mac is highly secure and reliable, making it ideal for high-quality user experience. Mac also supports cross-platform tools like Electron for browser-based games, but testing across devices is recommended to ensure cross-platform compatibility. |  |  | | --- | |  | | |  | | --- | | Strong support for web applications across different distributions using browsers like Firefox and Chrome. Linux is primarily used for server environments, but desktop distributions support full web functionality, enabling responsive web-based game experiences. Cross-platform testing is recommended due to variations in distributions. |  |  | | --- | |  | | |  | | --- | | High support for web applications and compatibility with multiple browsers (Chrome, Firefox, Edge). Windows has a broad user base, which can help attract more players. The development environment is user-friendly, and tools like Visual Studio enhance compatibility testing. Responsive design is necessary to ensure usability on different screen sizes. |  |  | | --- | |  | | |  | | --- | | Crucial for reaching a broad player base. Requires responsive design for varying screen sizes and interactions on Android and iOS. Mobile clients may require platform-specific optimizations, such as adjusting touch controls or simplifying interfaces for mobile use. Testing across devices is essential to ensure functionality and visual appeal on different mobile platforms. |  |  | | --- | |  | |
| **Development Tools** | |  | | --- | | Xcode for iOS, IntelliJ IDEA, and Java SDK for general development. Xcode has a free version but requires a developer license for publishing to iOS. The need for Mac devices for iOS testing could add to costs, and a dedicated team for Mac-specific development may be necessary. |  |  | | --- | |  | | |  | | --- | | Eclipse, IntelliJ IDEA, Java SDK for backend development, and cross-platform tools. Linux is open-source, so there are no licensing costs for the OS or development tools. Training may be required if team members are less familiar with Linux development environments, but Linux’s flexibility for backend and server-side development is a major advantage. |  |  | | --- | |  | | |  | | --- | | Visual Studio for Windows-specific builds, along with IntelliJ IDEA and Java SDK for web applications. Visual Studio offers strong integration with Windows but has licensing costs for some editions. Windows development is user-friendly, but expertise in Windows-specific configurations and cross-browser testing may be needed. |  |  | | --- | |  | | |  | | --- | | Android Studio for Android and Xcode for iOS are essential. The Java SDK allows for cross-platform development, though both iOS and Android development require specific environments (Mac for iOS, Android Studio for Android). Licensing costs are minimal but a dedicated team may be needed for cross-platform mobile expertise. | |

## 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**

Linux is a great choice here because it’s free, reliable, and very scalable. As an open-source platform, it doesn’t require licensing costs, and it’s super flexible, which makes it ideal for supporting lots of users. Plus, Linux works well with cloud environments, so Draw It or Lose It can easily scale as it grows.

1. **Operating Systems Architectures**

Linux’s setup is modular, which means it’s flexible enough for either a monolithic or a microservices architecture. This flexibility could be a plus for Draw It or Lose It, letting the app scale or adapt depending on future needs.

1. **Storage Management**

Cloud storage (like AWS or Google Cloud) is a solid option since it’s scalable and secure. Another benefit is redundancy—data can be replicated and spread across different servers, which means even if there’s a network hiccup, Draw It or Lose It won’t lose access to its images. This ensures smoother gameplay.

1. **Memory Management**

Linux handles memory well by using virtual memory and efficient paging. For Draw It or Lose It, this means the game can work with large image files and cache them without causing lags. It helps keep gameplay fast and smooth.

1. **Distributed Systems and Networks**

Using REST APIs with HTTPS is a good way to handle secure communication between devices. To make sure everything runs smoothly even with lots of players, using a microservices setup could help scale different parts of the game independently. Adding load balancing would keep

performance steady by spreading requests around and preventing slowdowns.

1. **Security**

Besides basic encryption and HTTPS, it’d be smart to add regular vulnerability checks and automated monitoring to boost security. For cross-platform security, using token-based authentication (like OAuth 2.0) helps keep user sessions secure on all devices.