

<Name-of-Software-Application>

# **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 | 01/26/25 | Donovan Taylor | Filled out the executive summary, design constraints, and the domain model sections. |

**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 current Android-only game, Draw It or Lose It, into a web-based distributed application that supports multiple platforms. The game is designed for teams competing to guess puzzles, with each team comprising multiple players. The software design focuses on achieving the following key objectives:

* Ensure game and team names are unique to prevent duplication and provide a seamless user experience.
* Implement a single-instance mechanism for the game in memory, ensuring a singleton pattern for centralized game management.
* Design the application to allow multiple teams and players, maintaining the structure and scalability required for future platform support.

This document outlines the design constraints, provides a UML diagram-based domain model explanation, and discusses object-oriented principles employed for an efficient and robust implementation. Adhering to these guidelines ensures the web-based application meets functional requirements and facilitates scalability and maintainability.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Developing a web-based, distributed game application introduces the following constraints:

1. Concurrency Management: The system must handle multiple concurrent users attempting to join teams, submit answers, or perform other game actions. A distributed environment increases complexity due to potential race conditions and data consistency issues.
   * Implication: Use of thread-safe data structures and synchronization mechanisms to ensure consistent state across all users.
2. Singleton Implementation: Only one instance of the game can exist in memory.
   * Implication: Implement a singleton design pattern in the GameService class to manage this constraint and provide centralized control of game state.
3. Unique Identifiers: Game, team, and player names must be unique, requiring efficient validation mechanisms.
   * Implication: The system must implement a validation process, potentially backed by a database or in-memory data structures, to ensure uniqueness without impacting performance.
4. Scalability and Platform Independence: The game should function seamlessly across different platforms, requiring adherence to cross-platform web technologies and distributed systems design principles.
   * Implication: Leverage cloud-based backend services, such as RESTful APIs, and a frontend compatible with modern web browsers.
5. Network Latency: A distributed environment might introduce latency in communication between clients and servers.
   * Implication: Optimize API responses, minimize data transmission, and provide fallback mechanisms for connection issues.

## [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)

Domain Model  
The provided UML diagram represents the Domain Model for the game application, with the following key components:

1. Entity Class:
   * A base class that encapsulates common attributes (id, name) and behaviors (getId(), getName(), toString()).
   * Principle: This demonstrates inheritance, allowing subclasses like Game, Team, and Player to share common functionality, reducing code redundancy.
2. GameService:
   * Serves as a singleton class to manage the game instance and operations like adding games or teams. It maintains lists of Game objects and unique counters (nextGameId, nextTeamId, nextPlayerId) for ID generation.
   * Principle: Implements the singleton pattern, ensuring only one instance of GameService exists.
3. Game Class:
   * Represents a game instance with a unique ID, name, and a list of teams. It allows the addition of new teams.
   * Principle: Encapsulation is used to ensure that teams can only be modified through controlled methods like addTeam().
4. Team Class:
   * Represents a team in a game, containing a unique ID, name, and a list of players. It provides methods to add players.
   * Principle: Composition is demonstrated as Team objects include Player objects, establishing a "has-a" relationship.
5. Player Class:
   * Represents a player with a unique ID and name.
6. ProgramDriver and SingletonTester:
   * ProgramDriver serves as the entry point (main()), while SingletonTester verifies the singleton implementation.

Relationships:

* GameService manages multiple Game objects (aggregation).
* Each Game manages multiple Team objects (composition).
* Each Team contains multiple Player objects (composition).

Object-Oriented Principles:

* Inheritance: Shared behavior in Entity is inherited by Game, Team, and Player.
* Encapsulation: Private attributes with public getters/setters in all classes.
* Composition: Teams and players are composed within games and teams, respectively.
* Singleton: Centralized game management in GameService.

"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 systems (macOS) offer a robust Unix-based environment that is well-suited for web-based server deployments. They benefit from strong stability and security, and many macOS server features are integrated into the operating system. However, macOS has a smaller market share for server deployments, and licensing or hardware costs may be higher compared to open-source alternatives. Overall, macOS can host web applications effectively, but may be best suited for smaller-scale deployments or development environments. > | Linux is widely recognized for its flexibility, scalability, and cost-effectiveness in server environments. Being open source, Linux distributions are free to use and offer extensive community support, making it ideal for hosting web applications that need to scale to thousands of players. While the learning curve may be steeper for administrators unfamiliar with Linux, its robust performance and customization capabilities make it a popular choice for production-grade web servers. | Windows Server provides a familiar and feature-rich environment for enterprises, with comprehensive management tools and integration with other Microsoft products. Although it offers excellent support and ease of use for administrators, Windows-based servers can incur significant licensing costs. Performance and stability have improved over time, yet some high-traffic web applications may benefit from the proven efficiency of Linux-based systems. | Mobile devices are generally not designed to host web-based applications in a production environment. Their hardware and operating systems are optimized for client-side tasks rather than continuous server processes. While mobile devices can run local servers for development or testing purposes, their limited resources, power constraints, and network variability make them unsuitable as primary hosts for a large-scale, web-based application. |
| **Client Side** | For Mac-based clients, a modern responsive HTML interface is supported by browsers like Safari, Chrome, or Firefox. Developers must ensure that the application is compatible with macOS web standards and design guidelines. The development process on macOS is streamlined by robust tools such as Xcode and IntelliJ IDEA, though testing across various browsers is essential to ensure consistent behavior. The cost and time investments are moderate, with a focus on optimizing user experience for desktop environments. | Linux desktop clients typically use browsers like Firefox and Chrome that adhere closely to web standards. Because Linux is largely open source and free, development costs are lower, and testing is required across different distributions and desktop environments to ensure consistency. Developers must balance performance and usability while accommodating the diversity in Linux setups, which may require additional expertise in troubleshooting environment-specific issues. | On Windows, client-side development must address a diverse range of web browsers including Edge, Chrome, and Firefox. The ecosystem is mature with many commercial and free development tools available, but additional testing is needed to account for legacy systems or varying user configurations. While the development costs can be higher due to licensing fees for certain tools, Windows provides extensive support for multimedia and accessibility, enhancing the overall user experience. | For mobile clients, ensuring compatibility across iOS and Android requires the development of a responsive, touch-optimized web interface. Developers may use frameworks like React, Angular, or cross-platform tools such as React Native and Flutter to streamline development. The process demands careful attention to differing screen sizes, performance optimizations, and OS-specific design guidelines. Although the initial development and testing phase can be time-intensive and require specialized expertise, the payoff is a versatile application that reaches a broad mobile audience. |
| **Development Tools** | Mac development commonly involves languages such as Java, Swift, and Objective-C, along with web technologies (HTML, CSS, JavaScript). IDEs like Xcode, IntelliJ IDEA, Eclipse, and Visual Studio Code are popular. Tools like Docker, Git, and various terminal utilities enhance productivity. The macOS environment supports a rich ecosystem for building and deploying web applications, though some proprietary tools may have licensing fees. | Linux offers a vast array of open-source development tools and languages including Java, Python, PHP, and Ruby. IDEs such as Eclipse, IntelliJ IDEA, NetBeans, and lightweight editors like Vim or Emacs are widely used. The strong emphasis on free and open-source software reduces licensing costs, though developers might need to navigate varying distributions and command-line tools. This environment encourages flexibility and cost efficiency in building scalable web applications. | Windows development leverages a mix of commercial and free tools. Languages such as C#, Java, and web-centric languages (HTML, CSS, JavaScript) are popular, with Visual Studio, IntelliJ IDEA, and Eclipse as common IDE choices. Windows also supports Docker and Git. However, licensing fees for commercial IDEs and certain development tools can increase costs. The platform offers extensive support and integration with other Microsoft services, which can be an advantage in certain enterprise scenarios. | Mobile development relies on specialized tools and languages: Android Studio with Java or Kotlin for Android, and Xcode with Swift or Objective-C for iOS. Cross-platform frameworks like React Native, Flutter, or Xamarin are also frequently used. These tools come with robust emulators, simulators, and testing frameworks to ensure compatibility across diverse mobile devices. While some toolchains may require paid licenses (e.g., the Apple Developer Program), many of the development tools for mobile are free or open source, though they do demand specific expertise and additional testing resources. |

## 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**: I suggest using a Linux server. Linux is well-known for its reliability, ability to handle many users, and low costs since it is free. This makes it a great choice for hosting Draw It or Lose It and supporting its expansion to different devices.
2. **Operating Systems Architectures**: Linux uses a design that keeps system tasks separate from user applications. This means the parts that manage hardware and system resources run independently from the applications. This design improves reliability and security. Additionally, Linux easily supports virtual environments and containers, which is useful for running a web-based game across multiple systems.
3. **Storage Management**: For storing game data on Linux, file systems like ext4 or XFS are good choices because they work quickly and keep data safe. For larger, distributed setups, systems such as Ceph or GlusterFS can spread the data across several machines. This ensures that game data and user information stay available and secure.
4. **Memory Management**: Linux is good at managing memory. It uses methods like virtual memory and caching to ensure that active applications have the memory they need, even when many users are connected. This helps keep the game running smoothly, even during busy times.
5. **Distributed Systems and Networks**: To allow the game to work on many different devices, the server can offer web services (RESTful APIs) that all clients (desktop, mobile, etc.) use to communicate. This setup means each part of the system works on its own while still talking to one another. To keep the service running smoothly, you can add tools like load balancing and backup systems to handle network problems.
6. **Security**: Security is very important. Linux comes with built-in features to help protect data. You can use encrypted connections (HTTPS) to secure data sent over the internet, and sensitive data on the server can be encrypted as well. While the initial setup might use a simple login system, you can upgrade to stronger methods like token-based logins in the future. Tools like SELinux or AppArmor add extra protection by controlling what software can do, and regular updates and security checks help keep the system safe.