

The Gaming Room

# **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 | 09/22/2024 | Zac Harrington | Changes have been made to the cover page, document revision history, executive summary, design constraints, system architecture view, domain model, and recommendations. |
| 2.0 | 10/6/2024 | Zac Harrington | Changes made to the evaluation and recommendations. |
| 3.0 | 10/20/2024 | Zac Harrington | Changes made to recommendations. |

**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 software design involves adapting an existing mobile game, Draw It or Lose It, into a web-based platform that can be accessed across multiple devices. The solution is to develop the game in a distributed web-based environment that meets the client's requirements for game teams, unique identifiers for teams and players, and game state management. The design will ensure that the software is scalable, supports multi-user interactions, and adheres to best practices in object-oriented programming. A detailed approach involving design patterns like Singleton and Iterator will be used to maintain system efficiency and reliability.

## Requirements

Business Requirements: The system must support team-based gameplay with unique identifiers for teams and players. It must handle game state transitions smoothly and offer cross-platform access to users.

Technical Requirements: Implement Singleton pattern to ensure only one instance of game service. Use the Iterator pattern for managing the team and player creation, ensuring that names are unique across all instances.

## [Design Constraints](#_2et92p0)

Developing the game for a web-based distributed environment requires considering factors such as:

* Scalability: The system should handle multiple concurrent users without performance degradation.
* Consistency: Game states must remain consistent across different devices and sessions.
* Data Storage: Storing unique player and team information and ensuring that no duplicates exist.
* Security: User data, including player and team information, must be protected against unauthorized access.

## [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 demonstrates how the system's classes interact:

* Entity: A base class that holds shared attributes like id and name, representing objects like games, teams, and players.
* GameService: Implements the Singleton pattern, ensuring only one instance of the service exists. It also manages the creation of game, team, and player objects.
* Game, Team, and Player: Derived from the Entity class, these represent core elements of the game. The Iterator pattern is used in the addTeam() and addPlayer() methods to ensure that each team and player name is unique before adding them to the system.
* ProgramDriver: Used to initiate and run the program, simulating the game environment for testing purposes.

**"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, while stable and user-friendly, isn't commonly used as a server platform. It's more common for web development on client machines than as a web server. However, it's Unix-based, offering many benefits like strong security and good performance. The downside is that fewer enterprise server tools are optimized for MacOS compared to Linux. | Linux is widely known for its performance, stability, and strong security, making it ideal for web servers. It’s open-source and customizable, allowing for efficient server setups. Linux also has a large ecosystem of server tools and is highly preferred for scalable web-based applications. | Windows servers are commonly used for enterprise solutions and integrate well with Microsoft services. However, they can be more resource-intensive compared to Linux, and licensing costs are higher. Windows Server is also compatible with many industry-standard tools but lacks the flexibility of Linux. | Mobile devices are not traditionally used for hosting web-based software applications. However, they are critical in consuming and interacting with web applications, especially in hybrid or responsive design environments. |
| **Client Side** | MacOS is widely used in client-side web development, especially in creative industries. The cost can be higher, but its Unix foundation makes it great for development. However, fewer software tools are available compared to Windows. | Linux is highly customizable and lightweight, making it a great option for developers, though less so for general users. It is secure and has many development tools, but its user interface may not be as user-friendly, requiring more expertise. | Windows is the most commonly used OS in client environments, offering broad compatibility with software. It’s user-friendly and supports a wide range of development tools. However, security concerns and higher resource use can be a downside. | Mobile devices provide the flexibility of accessing applications on the go, but their hardware limitations and the mobile OS can restrict development and user interaction capabilities, requiring the use of mobile-optimized frameworks. |
| **Development Tools** | MacOS provides access to many development tools (Xcode, Homebrew, etc.), making it ideal for software development, particularly in iOS applications. However, it is less common for web development outside of specific industries. | Linux has a vast number of development tools, especially for web applications. Its open-source nature means there's flexibility in choosing IDEs, languages, and databases. Popular tools like Eclipse, IntelliJ, and Apache work well with Linux. | Windows has access to a large number of IDEs (Visual Studio, IntelliJ, etc.), making it very flexible for developers. It's well-suited for enterprise applications and development but comes with licensing costs and security concerns. | Mobile devices are not commonly used for software development directly but are the target platform for many web-based applications. Development typically happens on traditional platforms, but frameworks like Cordova allow developers to build cross-platform mobile apps. |

## 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: To allow The Gaming Room to expand Draw It or Lose It to other computing environments, I recommend using Linux as the operating platform for the web server. Linux is well-known for its high performance, stability, and robust security features. Additionally, its open-source nature makes it cost-effective and flexible, allowing developers to customize the system as needed. Given the web-based nature of the application, Linux’s compatibility with cloud services and web servers like Apache and Nginx ensures a seamless deployment process across distributed environments.
2. Operating Systems Architectures: The recommended system architecture for the Linux-based platform should include Apache or Nginx as the web server to manage HTTP requests efficiently. Additionally, using Docker for containerization will help streamline the development process by creating isolated environments for application deployment. Furthermore, MySQL or PostgreSQL should be used for data storage to ensure the reliability of data management. This architecture will allow for scalable deployments, efficient use of system resources, and compatibility with modern web frameworks.
3. Storage Management: For storing images, game states, and other relevant data, I recommend utilizing cloud storage services such as Amazon S3 or Google Cloud Storage. These solutions are scalable, secure, and compatible with Linux-based environments. They offer flexible storage options and easy access for developers. Cloud storage integrates well with the overall system architecture, allowing the game to store and retrieve assets quickly and efficiently. It also ensures that The Gaming Room can scale up as user demand increases without worrying about storage limitations.
4. Memory Management: Linux offers excellent memory management features, including caching and load balancing. Implementing load balancing will ensure even distribution of system resources, allowing the application to handle a large number of users simultaneously. Caching frequently accessed game assets, such as images and metadata, will also improve system performance by reducing the number of requests made to the database. Tools like Memcached or Redis can be integrated to manage cache effectively.
5. Distributed Systems and Networks: Since Draw It or Lose It needs to run across multiple platforms and communicate between them, I recommend leveraging distributed systems using cloud services like AWS or Google Cloud. The use of a distributed database such as MongoDB or Cassandra will allow the application to scale horizontally, meaning that as more users join, additional database nodes can be added to accommodate the increased traffic. Additionally, RESTful APIs will enable communication between different platforms and devices in a consistent and secure manner, while Content Delivery Networks (CDNs) will optimize the delivery of game assets globally.
6. Security: Protecting user data is critical for the client, and Linux’s inherent security features make it an ideal choice. The recommended security measures include using OAuth or JWT for user authentication, ensuring secure access to the application. Additionally, implementing SSL/TLS encryption will protect data in transit between the client and server. Regular security patches and updates should be scheduled to ensure that vulnerabilities are addressed promptly. By using Linux's strong permission model and firewalls like iptables, unauthorized access to the system can be prevented, further securing sensitive user data.