

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 4**

**Domain Model 4**

**Evaluation 5-7**

**Recommendations 8-9**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/24/2024 | Hector Banos Ramos | I wrote an initial version of the software design document, outlining the executive summary, design constraints, domain model analysis. |
| 1.1 | 02/06/2024 | Hector Banos Ramos | I completed the evaluation table, and addressed the server side, client side and development tools sections. |
| 1.2 | 02/19/2024 | Hector Banos Ramos | I proposed recommendations for the project addressing factors like operating system architectures, memory and storage management, distributed systems and networks and security. |

**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 aims to expand its current Android app, Draw It or Lose It, to a web-based format. The software design solution involves creating a robust web application that supports multiple teams and players. The use of software design patterns, such as Singleton and Iterator, ensures efficient memory management and uniqueness of game and team names. This document outlines the design constraints, UML class diagram, and recommendations for operating platforms and system architectures.

## Requirements

The project requirements for Draw It or Lose It encompass both business and technical aspects. From a business perspective, the client expects the game to expand onto various operating systems, transitioning from an Android app to a web-based format. This expansion aims to increase accessibility and reach for players. Additionally, the game should support multiple teams and players, allowing for engaging multiplayer experiences.

From a technical standpoint, the system must ensure efficient memory and storage management to optimize performance and accommodate large volumes of data. Seamless communication between different platforms is vital, requiring the implementation of distributed software architectures and robust network connectivity. Lastly, the system must prioritize security measures to protect user information across platforms, including encryption and access control mechanisms.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment imposes the following constraints on the system architecture:

Network Latency: In a web-based environment, the application must handle potential network latency. This impacts real-time interactions, requiring strategies like asynchronous processing to maintain a responsive user experience.

Browser Compatibility: Web applications need to support various browsers with different capabilities; thus, compatibility is essential to ensure a consistent user experience across different browsers.

Data Security: Web-based applications are exposed to potential security threats. Implementing secure communication protocols (HTTPS), data encryption, and user authentication mechanisms is crucial to protect sensitive user information.

Scalability: As the application may have a growing user base, scalability is a key concern. Designing for scalability involves utilizing scalable database solutions and optimizing algorithms to handle increasing data and user loads.

Cross-Platform Compatibility: The web-based application needs to be compatible with various devices and screen sizes.

In summary, the application must consider factors such as network latency, browser compatibility, and data security. The use of software design patterns ensures a scalable and efficient solution.

## [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 represents the Domain Model, showcasing classes like Game, Team, Player, and GameService. The relationships demonstrate object-oriented principles, such as inheritance and composition. The Singleton pattern in GameService ensures a single instance of the class, and the Iterator pattern is used to manage game and team uniqueness efficiently.

The UML class diagram show the following key classes and their relationships:

Entity: The base class holding common attributes and behaviors like id and name.

GameService: Implemented as a Singleton, managing the creation and retrieval of Game instances, and demonstrates the Singleton pattern.

Game: Represents a game instance, connected to Team instances. Utilizes composition, demonstrating object-oriented principles.

Team: Represents a team within a game, connected to Player instances.

Player: Represents a player within a team.

The Singleton pattern implemented in GameService ensures a singular, global access point to the GameService instance, and that is a critical aspect for effective management of all games. Moreover, the composition principle is exemplified as the Game class is comprised of Team instances, and Team is, in turn, composed of Player instances. Additionally, the classes exhibit associations, meaning relationships, for example, GameService is associated with Game, Game with Team, and Team with Player, those relationships show the interdependencies within the game application entities. Also, the common base class Entity implies an inheritance relationship, as both Game and Team inherit shared attributes and behaviors. Lastly, the application of the Iterator pattern is evident in GameService's methods, like getGameCount, facilitating efficient iteration over collections of games.

**"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** | Server-Side Mac provides solid hosting capabilities for web-based software applications. Its Unix-based architecture ensures stability and security, making it a reliable choice for hosting. However, considerations for cost include licensing fees for development tools and potential additional resources required for cross-platform development. Mac may have higher licensing costs compared to Linux and Windows. | Linux is widely used for hosting web-based applications due to its stability, security, and cost-effectiveness (typically open-source). It offers various server distributions tailored for different needs, making it highly versatile. Linux server distributions are typically open-source and freely available. However, there may still be costs related to support services, maintenance, or enterprise-level distributions | Windows offers server-based deployment options for hosting web applications. Windows server operating systems often involve licensing costs, which can vary depending on the edition and the number of server instances. While it may have higher licensing costs compared to Linux, it provides good compatibility with Microsoft technologies and may be preferred for environments where integration with Microsoft products is important. | Mobile devices, including iOS and Android, can host web-based software applications.  For mobile devices, including iOS and Android, there are no direct licensing costs for the server operating system.  However, development considerations include adapting the application for different screen sizes, input methods, and performance optimization. |
| **Client Side** | Client Side Supporting multiple types of clients on Mac requires expertise in cross-platform development frameworks like React Native or Xamarin. Development tools like Xcode are essential for building and testing applications on MacOS. Time may be impacted by the need for platform-specific optimizations and testing.  Development tools on Mac may require upfront licensing costs, but open-source alternatives are available. | Linux-based clients require consideration for compatibility with various desktop environments and libraries. Development on Linux can be cost-effective due to open-source tools and libraries, but expertise may be needed to ensure compatibility across distributions. Time may be affected by the need for testing on multiple Linux distributions.  Many development tools on Linux are open source, reducing upfront costs, but expertise may be required for setup and configuration. | Windows clients can benefit from development using Visual Studio and .NET frameworks. Compatibility with Windows versions and hardware configurations should be considered.  Windows Server environments often require less expertise for setup and management, but licensing costs can be a significant factor.  Windows may have licensing costs, but there are free and open-source alternatives available. | Developing for mobile devices involves using platform-specific languages (Swift for iOS, Java/Kotlin for Android) and development tools (Xcode, Android Studio). Cross-platform frameworks like Flutter or React Native can streamline development for multiple mobile platforms. In general, developing mobile devices may require additional time and expertise compared to desktop platforms.  Development tools for mobile platforms may have upfront licensing costs, but open-source alternatives are available. Time and expertise required for mobile development may vary depending on familiarity with the tools and languages used. |
| **Development Tools** | Xcode is the primary development tool for building software on MacOS, supporting languages like Swift and Objective-C. This could influence the composition of the team, potentially requiring members with specialized knowledge in macOS development. Additionally, IDEs like Visual Studio Code and JetBrains IDEs are popular among Mac developers.  Also, there are potential licensing costs related to Mac development tools. Xcode, the primary development tool for macOS, is available for free on the Mac App Store. However, if the project requires additional development tools or plugins, there may be costs. | Linux development involves using a variety of programming languages such as C, C++, Python, and Java, with development tools like GCC, Python IDEs, and Eclipse. Text editors like Vim and Emacs are also widely used. Those requirements may require a development team with diverse skill sets capable of working with multiple programming languages and tools. When it comes to costs Linux development tools are primarily open-source, and typically do not have direct licensing costs. However, there may be costs associated with optional tools or support services, particularly for proprietary IDEs like JetBrains IDEs. | Windows development commonly involves using programming languages like C#, C++, and .NET frameworks, with IDEs like Visual Studio. Other tools like JetBrains IDEs and Notepad++ are also used. This may require a development team with expertise in Windows-specific technologies. Talking about costs, there are potential licensing costs related to Windows development tools. Proprietary IDEs like Visual Studio may require purchasing licenses for developers, and there could be additional costs for specialized tools or plugins. | Mobile development on iOS requires using Xcode and Swift/Objective-C. Android development involves using Android Studio and Java/Kotlin. Developing for mobile devices, including iOS and Android, requires expertise in platform-specific technologies mentioned previously.  Considering the complexity of these platforms the project may need specialized development teams or integration of mobile development expertise into broader development teams. There are potential licensing costs related to mobile development tools like costs associated with obtaining Apple/Google developer accounts for distribution. |

## 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 expand the Draw It or Lose It app to other platforms, I recommend leveraging a Linux-based operating platform. Linux offers robust hosting capabilities for web-based applications, characterized by stability, security, and cost-effectiveness. Its open-source nature provides versatility, with various server distributions designed to cover different needs. While Linux may require expertise to ensure compatibility across distributions, its cost-effectiveness and reliability make it an ideal choice for accommodating the game's expansion.
2. **Operating Systems Architectures**: The chosen Linux operating platform uses a monolithic kernel architecture, characterized by a single, large kernel that handles all system activities. This architecture ensures efficient resource management and performance optimization, crucial for hosting web-based applications like Draw It or Lose It. Additionally, Linux supports a modular design, allowing for the addition or removal of kernel modules dynamically, enhancing scalability and customization.
3. **Storage Management**: To complement the Linux operating platform, I recommend using cloud storage solutions such as Amazon S3 or Google Cloud Storage for hosting the image data required for Draw It or Lose It. Cloud storage offers several advantages over traditional file systems. Firstly, it provides virtually unlimited scalability, allowing the application to store and retrieve large volumes of data without worrying about storage capacity constraints. Additionally, cloud storage services offer built-in redundancy and data replication across multiple geographic regions, ensuring high availability and data durability. By utilizing cloud storage, Draw It or Lose It can benefit from seamless scalability, high availability, and robust security, enhancing the overall reliability and performance of the application.
4. **Memory Management**: The recommended Linux operating platform employs various memory management techniques to optimize the performance of Draw It or Lose It. Techniques such as demand paging, virtual memory, and memory compression ensure efficient memory allocation and utilization, enhancing system responsiveness and stability. Additionally, Linux provides mechanisms to handle memory overload situations efficiently, preventing system crashes and maintaining overall reliability. For example, one well-known mechanism in Linux for handling memory overload is the OOM (Out-of-Memory) killer. When the system runs out of memory and essential processes are at risk, the OOM killer identifies and terminates non-essential processes to free up memory, ensuring system stability.
5. **Distributed Systems and Networks**: To facilitate communication between various platforms, Draw It or Lose It can use a distributed software architecture based on RESTful APIs (Representational State Transfer). RESTful APIs allow seamless interaction between different components of the application, enabling data exchange and synchronization across platforms. By adhering to REST principles, such as statelessness and uniform interfaces, the application can achieve interoperability and scalability, crucial for accommodating multiple platforms. Additionally, implementing a reliable network infrastructure with redundant connectivity is essential to ensure continuous communication between devices. Redundant connections help mitigate the impact of connectivity issues or outages, maintaining uninterrupted access to the game for users across different platforms. Also, considering dependencies within the distributed systems and networks, measures such as load balancing and fault tolerance can enhance system reliability and performance, ensuring smooth gameplay experiences for players.
6. **Security**: Security is crucial for protecting user information on and between various platforms. The recommended Linux operating platform offers robust security capabilities, including built-in firewall management and access control mechanisms. Implementing HTTPS encryption for data transmission ensures secure communication between clients and servers, safeguarding sensitive user data from unauthorized access or interception. Additionally, regular security updates and patches from the Linux community help mitigate potential vulnerabilities and ensure ongoing protection against emerging threats.