

# 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 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

**Recommendations 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/18/24 | Duane Wegner | Initial version of the software design document |
| 1.1 | 10/02/24 | Duane Wegner | Updated The Evaluation section of the software design document |
| 1.2 | 10/15/24 | Duane Wegner | Updated the Recommendations sections of the software design document |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has requested the design of a web-based version of their existing game, Draw It or Lose It. The game allows multiple players to compete by guessing drawings within a time limit. The goal is to expand this game to various platforms, ensuring seamless communication between different devices in a distributed environment. This document outlines the solution by addressing the software design requirements and design constraints to develop a scalable, secure, and high-performing application. We propose using a cloud-based architecture to facilitate real-time communication and data management across platforms like Windows, Linux, Mac, and mobile devices. This approach will also support scalability as more users join the game over time.

Critical aspects include implementing secure user authentication, robust memory and storage management, and ensuring consistent performance across all devices. This document will guide the client through the technical solution, design constraints, and platform considerations for successful development.

## Requirements

The client’s business requirements include expanding the game Draw It or Lose It to multiple platforms while maintaining performance and ensuring security. The technical requirements include ensuring that the application can scale to accommodate more players, work in real-time, and communicate across various platforms in a distributed environment.

## [Design Constraints](#_2et92p0)

Design constraints for this project include ensuring that the game works on a variety of platforms, managing network latency for real-time interactions, scaling the application to handle growing numbers of users, and maintaining security in a distributed environment. The game must function efficiently on platforms with varying hardware capabilities, particularly mobile devices that have limited memory and processing power. Responsive communication between players, whether on desktop or mobile, is crucial to the gaming experience.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram for the game shows a clear structure, with classes such as Player, GameSession, Drawing, and Guess representing the core components of the system. These classes follow object-oriented programming principles like encapsulation and inheritance, which improve the system's modularity and make it easier to maintain. The relationships between these classes ensure smooth communication between game sessions and players while optimizing performance and efficiency.

**"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 offers good performance and a solid security framework but is expensive in terms of hardware. It's suitable for smaller deployments or teams familiar with Apple’s ecosystem, but scalability and customization options are limited compared to other platforms. | Linux is highly scalable, cost-effective, and offers excellent security features, making it ideal for hosting the game server. Its open-source nature allows for deep customization and flexibility in server configurations. It supports various server environments like Apache and Nginx. | Windows servers provide strong enterprise solutions, especially for .NET-based applications, but are more expensive. Licensing fees for Windows Server editions can increase costs. Windows is less flexible and scalable than Linux but is widely used in enterprises. | Hosting a server on mobile devices is not feasible due to resource limitations. However, mobile devices will access the game via web interfaces or native apps that communicate with a back-end server. |
| **Client Side** | Development for Mac clients require expertise in Objective-C or Swift for native apps. Users primarily access the game through web browsers like Safari, and native applications require adherence to Apple's development guidelines. | Linux users will mostly access the game through web browsers like Firefox or Chrome. Native Linux apps are less common due to a smaller user base, but responsive web design ensures cross-platform compatibility. | Windows is the dominant desktop OS, making it a key platform for desktop clients. Developing for Windows requires C#, .NET, or C++. Windows users will also access the game through web browsers like Edge or Chrome. | Mobile devices (iOS and Android) will access the game via native apps built using frameworks like React Native or Xamarin. Resource optimization is crucial due to limited processing power and memory on mobile platforms. |
| **Development Tools** | Xcode is the primary development environment for MacOS, using Swift and Objective-C. Cross-platform frameworks like Unity can also be used for both mobile and desktop apps. | Development on Linux can be done with tools like Eclipse or Visual Studio Code, with languages such as Java, C++, or Python. There are no licensing costs for these development tools as Linux is open-source. | Visual Studio is the primary IDE for Windows, supporting C# and .NET. Licensing fees for Visual Studio Professional or Enterprise editions might impact the budget, though it offers robust support for Windows app development. | Mobile app development requires tools like React Native, Xamarin, and Android Studio, enabling cross-platform compatibility between Android and iOS. Licensing costs for development tools are minimal with open-source frameworks, but some premium tools might incur fees. |

## 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**: Based on the analysis, Linux is the recommended operating platform for hosting the game server. It offers scalability, cost-efficiency, and strong security features, making it the most suitable choice for expanding *Draw It or Lose It* across multiple platforms.
2. **Operating Systems Architectures**: The Linux architecture allows for efficient management of server resources, with an open-source nature that enables customization. It supports cloud-based services and provides excellent uptime and security, key elements for a web-based gaming platform.
3. **Storage Management**: A cloud storage solution such as AWS S3 or Google Cloud Storage is recommended to handle data from the game. These services are scalable and secure, ensuring the game can store large amounts of user and game session data reliably.
4. **Memory Management**: Linux’s memory management techniques, such as caching and paging, will support multiple concurrent game sessions. This is especially important in a real-time game where performance is critical.
5. **Distributed Systems and Networks**: WebSockets or HTTP/2 will facilitate real-time communication between clients and the server. These protocols ensure low-latency data exchange, allowing players to interact with the game seamlessly across platforms. The distributed system will manage dependencies between components, ensuring smooth gameplay despite connectivity issues.
6. **Security**: To protect user information, SSL/TLS encryption will be employed for secure communication across platforms. Additionally, using OAuth 2.0 for user authentication will ensure robust security. Each platform will have data encryption mechanisms in place to safeguard sensitive user data both in transit and at rest.