

Draw It or Lose It

# **CS 230 Project Software Design**

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 | 05/17/2023 | Gabriel Brands | Summary, requirements, design constraints, system architecture review, domain model, evaluation, 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)

This software design document outlines the design approach for a web-based distributed game application. The objective of this document is to provide a clear understanding of the software design problem and present a comprehensive solution to meet the client's requirements. The game application aims to provide an engaging and interactive gaming experience to users across different platforms. The proposed solution leverages a web-based distributed environment, allowing users to access the game seamlessly through web browsers on various devices. To achieve this, the software design will incorporate modern web technologies, such as HTML, CSS, and JavaScript, along with server-side components to manage game logic and data storage. The design will emphasize scalability, performance, and usability to deliver a high-quality gaming experience

## [Design Constraints](#_2et92p0)

Network Latency: As the game application operates in a distributed environment, network latency can impact real-time interaction and responsiveness. To mitigate this, the design should incorporate efficient data transfer protocols and minimize the number of network round trips.

Browser Compatibility: The game application should be compatible with a wide range of web browsers to ensure a consistent user experience across different platforms. This requires careful consideration of browser-specific features, standards compliance, and graceful degradation for older browser versions.

Security: The distributed nature of the application introduces security concerns. Measures need to be implemented to protect user data, prevent unauthorized access, and ensure secure communication between the client and server components. This includes encryption of sensitive information, robust authentication mechanisms, and input validation.

Scalability: The game application should be designed to accommodate a growing user base and handle increased traffic. This involves utilizing scalable infrastructure, implementing load balancing techniques, and employing caching mechanisms to optimize performance.

Data Consistency: In a distributed environment, maintaining data consistency across multiple servers can be challenging. The design should incorporate techniques such as distributed databases, replication, and synchronization to ensure consistent data access and integrity.

User Interface Design: The user interface (UI) of the game application should be intuitive, visually appealing, and responsive. It should provide clear feedback to users, guide them through the gameplay, and support different screen sizes and resolutions.

Adhering to these design constraints will ensure the successful development of the game application in a web-based distributed environment, providing an enjoyable user experience while addressing potential challenges and limitations.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram represents the Domain Model for the game application. It provides an overview of the main classes and their relationships, showcasing the key entities and behaviors of the system. The following classes are depicted in the diagram:

Entity: This is a base class that holds common attributes and behaviors shared by other classes in the domain model. It demonstrates the object-oriented programming principle of inheritance, allowing other classes to inherit its properties and methods.

User: This class represents a user of the game application. It encapsulates user-specific information, such as username, password, and game statistics. The User class demonstrates the encapsulation principle by encapsulating data and providing methods to access and manipulate it securely.

Game: This class represents an instance of a game within the application. It contains information about the game's state, players, and game-specific data. The Game class demonstrates the aggregation principle by aggregating multiple Player objects.

Player: This class represents a player participating in a game. It holds player-specific information, such as score and current game status. The Player class demonstrates the association principle, showing the relationship between players and games.

Leaderboard: This class represents the leaderboard of top players in the game application. It tracks and displays

**"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/**  **Client side** | Mac can be used for web application hosting, but it is more commonly used for desktop development.  It offers a Unix-based environment, which can provide similar advantages to Linux in terms of server capabilities.  Advantages:  Mac provides a user-friendly and intuitive interface, which can make development and deployment processes smoother.  It has a strong ecosystem of development tools and frameworks that are well-suited for web development.  Mac systems offer good stability and security. Weaknesses:  Mac may have higher licensing costs compared to Linux for server operating systems.  It may not be as widely used for server deployments as Linux or Windows. | Linux offers robust server-based deployment options and is widely used for hosting web applications.  It provides excellent stability, security, and scalability for server configurations.  Linux has a vast array of open-source tools and libraries that are well-suited for web development.  Advantages:  Linux is highly customizable and can be tailored to specific server requirements.  It has a large and active community that provides continuous support and frequent updates.  Licensing costs for Linux operating systems are generally lower compared to proprietary solutions.  Weaknesses:  Linux may require more technical expertise to configure and manage compared to other operating platforms.  The user interface may be less user-friendly for those less familiar with command-line interfaces.  . | Windows is commonly used for hosting web applications and provides a variety of server deployment options.  It offers a user-friendly interface and has extensive support for web development frameworks and tools.  Advantages:  Windows provides seamless integration with other Microsoft technologies, making it suitable for organizations already using Microsoft products.  It offers good compatibility with popular web development frameworks and tools.  Windows has a wide range of server management and monitoring tools available.  Weaknesses:  Licensing costs for Windows server operating systems can be higher compared to Linux.  Windows servers may require more frequent updates and maintenance compared to Linux. | To support multiple client types (iOS, Android, and web browsers), a hybrid or cross-platform development approach can be used.  Hybrid frameworks like React Native or Flutter allow developers to write code once and deploy it across different platforms.  Web-based applications can be developed using responsive design techniques to ensure compatibility across various web browsers and screen sizes.  Development teams need to have expertise in the chosen hybrid or cross-platform framework and web development technologies. |
| **Development Tools** | * Relevant programming languages for web-based application development include HTML, CSS, and JavaScript. * IDEs like Visual Studio Code, WebStorm, or Atom are commonly used for web development. * Frameworks like React, Angular, or Vue.js are popular choices for building web-based applications. * Mobile application development may require knowledge of native languages like Swift for iOS and Java or Kotlin for Android. * Hybrid frameworks like React Native or Flutter provide cross-platform development capabilities. | | | |

## 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 Draw It or Lose It to other computing environments, it is recommended to use a cloud-based operating platform. Cloud platforms provide a centralized infrastructure that can accommodate the game's expansion and handle increased user demand. They offer scalability and flexibility by allowing applications to be deployed across multiple operating systems and devices. Examples of cloud platforms include Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

* To expand Draw It or Lose It to other computing environments, it is recommended to use a cloud-based operating platform. Cloud platforms offer a range of benefits that make them suitable for hosting and deploying applications across multiple operating systems.
* Cloud platforms provide scalability, allowing the game to accommodate increased user demand and handle growing workloads. With cloud services, you can easily scale up or down the resources allocated to the game based on its requirements. This ensures that Draw It or Lose It can handle peak usage periods without performance issues.
* Flexibility is another advantage of cloud platforms. They support various operating systems, enabling you to deploy the game on different environments seamlessly. Whether you want to target Windows, macOS, Linux, Android, or iOS, a cloud-based operating platform can provide the necessary infrastructure and support.
* By utilizing a cloud platform, The Gaming Room can benefit from centralized management and administration. The platform takes care of hardware and infrastructure management, allowing the developers to focus on enhancing the game and delivering new features. Cloud platforms also provide built-in services and tools that simplify tasks such as scaling, load balancing, and monitoring.
* Additionally, cloud platforms offer high availability and reliability. They typically have multiple data centers distributed geographically, ensuring redundancy and minimizing the impact of potential hardware failures or network outages. This results in improved uptime for Draw It or Lose It, reducing downtime and enhancing the gaming experience for users.
* Overall, a cloud-based operating platform provides the necessary infrastructure, scalability, flexibility, and reliability to expand Draw It or Lose It to various computing environments and operating systems. It simplifies management and ensures the game can handle increasing user demand while maintaining optimal performance.

1. **Operating Systems Architectures**: The chosen cloud-based operating platform should support various architectures to cater to different operating systems. Two commonly used architectures are x86 and ARM.

* **x86 Architecture:** The x86 architecture is widely supported and compatible with most operating systems, including Windows, macOS, and Linux. It is based on the Intel x86 processor architecture and provides a standard instruction set architecture.
* **ARM Architecture:** The ARM architecture is commonly found in mobile devices and offers energy efficiency. It is compatible with operating systems such as Android and iOS. ARM-based processors are designed for low-power consumption and are prevalent in smartphones, tablets, and other mobile devices.
* By selecting a cloud platform that supports both x86 and ARM architectures, you can ensure compatibility with a wide range of operating systems.

1. **Storage Management**: An appropriate storage management system for the recommended operating platform is a distributed file system. Distributed file systems allow data to be stored and accessed across multiple servers or nodes, providing improved performance, fault tolerance, and scalability. Two popular options are Hadoop Distributed File System (HDFS) and Amazon S3.

* **Hadoop Distributed File System (HDFS):** HDFS is designed for storing and processing large datasets. It is highly scalable and fault-tolerant, as data is distributed across multiple nodes in a cluster. HDFS can handle the storage requirements of a game like Draw It or Lose It, which may involve large amounts of user-generated content or game assets.
* **Amazon S3:** Amazon Simple Storage Service (S3) is a cloud-based object storage service. It provides durability, availability, and scalability for storing and retrieving data. S3 is well-suited for storing game-related assets, user data, and other files required by Draw It or Lose It.
* By utilizing a distributed file system like HDFS or Amazon S3, you can ensure efficient storage management, improved performance, and the ability to handle increasing data volumes as the game expands.

1. **Memory Management**: The recommended operating platform should employ effective memory management techniques to ensure optimal performance of the Draw It or Lose It software.

* **Virtual Memory:** Virtual memory is a memory management technique that allows the operating system to manage and allocate memory resources efficiently. It enables the game to access a larger address space than the physical memory available. Virtual memory allows for multitasking and improves performance by moving data between physical memory and disk storage as needed.
* **Caching:** Caching is another memory management technique that can enhance performance. By storing frequently accessed data in faster memory, such as RAM or solid-state drives (SSDs), caching reduces the need for disk accesses, which are typically slower. This can result in faster data retrieval and improved overall performance for the game.
* The recommended operating platform should incorporate virtual memory management and caching mechanisms to optimize memory usage and enhance the performance of Draw It or Lose It.

1. **Distributed Systems and Networks**: To enable communication between various platforms and devices, a distributed systems approach can be adopted. This involves breaking down the game into distributed components that communicate with each other over a network.

* **API Integration:** Implementing a well-defined Application Programming Interface (API) is crucial for enabling communication between different platforms. An API specifies how different software components should interact and exchange data securely. By designing and implementing a robust API, Draw It or Lose It can communicate with other platforms and devices seamlessly.
* **Message Queuing:** Utilizing a message queuing system, such as RabbitMQ or Apache Kafka, can facilitate asynchronous communication between components in a distributed environment. In a distributed system, components may not always be available or connected. Message queuing allows the game to send and receive messages reliably, even if some components are offline or experiencing connectivity issues. This ensures the game's functionality remains intact, even during network outages or intermittent connectivity.
* By implementing API integration and utilizing message queuing systems, Draw It or Lose It can communicate effectively between various platforms, devices, and components within a distributed system.

1. **Security**: Ensuring the security of user information on and between various platforms is crucial for protecting user privacy and maintaining user trust. The recommended operating platform should provide robust security features and capabilities. Here are some key security measures to consider:

* **User Authentication and Authorization:** Implement a robust authentication mechanism to verify the identity of users and ensure they have the appropriate access privileges. This can include techniques like username/password authentication, multi-factor authentication (MFA), or integration with third-party authentication providers. Authorization mechanisms should be in place to control access to sensitive data and features within the game.
* **Encryption:** Utilize encryption techniques to protect sensitive user data both during storage and transmission. Secure Socket Layer/Transport Layer Security (SSL/TLS) encryption can be used to encrypt network communications between different platforms and devices. Encryption algorithms like Advanced Encryption Standard (AES) can be employed to encrypt data at rest, ensuring that even if the data is compromised, it remains unreadable without the decryption key.
* **Secure APIs:** Implement secure APIs with proper authentication and access control mechanisms. This can involve using industry-standard protocols like OAuth 2.0 to authorize API requests and ensure that data exchanged between different platforms remains protected. Care should be taken to validate and sanitize user input to prevent common security vulnerabilities like injection attacks.
* **Regular Security Audits:** Conduct periodic security audits and vulnerability assessments to identify and address any potential security weaknesses in the system. Stay updated with security patches and best practices provided by the operating platform provider. Regularly review and update security measures to protect against emerging threats and vulnerabilities.
* By incorporating these security measures and following best practices, Draw It or Lose It can ensure the protection of user information on and between various platforms, maintaining a secure gaming environment.