

Draw It or Lose It

# **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 | 11/07/2024 | Harrison Labrecque | Adding details related to the software applications in relation to business requirements and technical requirements. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room, a client of Creative Technology Solutions (CTS), has engaged our services to develop a web-based version of their widely popular mobile game, Draw It or Lose It. Currently, the game is exclusively available on the Android platform. It features teams of players who compete to guess a puzzle based on images that are progressively drawn over a 60-second interval. The objective of the web-based version is to faithfully replicate the core gameplay experience while integrating the necessary features and scalability to support a multi-team, multi-player environment.

Our proposed solution entails the creation of a web application specifically designed to accommodate multiple teams, with each team comprising several players. The game will consist of four rounds, during which progressively rendered images will serve as clues for the puzzles. To facilitate a seamless gaming experience, each team, player, and game will be assigned a unique identifier, thereby preventing name conflicts and ensuring effective tracking of participants. Real-time updates regarding drawing progress, guesses, and overall game state will be implemented through the use of WebSockets, a robust communication system that ensures reliable and secure communication between players and the server.

This strategic approach will provide users with an interactive, real-time gaming experience, enabling dynamic gameplay while mitigating potential issues such as duplicate names, unsynchronized game states, and unauthorized access. By leveraging contemporary web technologies, including React, Node.js, and WebSocket, we aim to develop a secure, scalable, and efficient system that will be straightforward to maintain and enhance in the future. This document outlines the essential design considerations and technical constraints integral to the successful implementation of the Draw It or Lose It web application.

## Requirements

The "Draw It or Lose It" web game is required to meet several critical business and technical specifications to ensure optimal functionality. Primarily, the game must support the simultaneous participation of multiple teams within a single game instance. Each team will comprise several players, necessitating real-time interaction among all participants. This requirement calls for a system wherein teams, players, and games are assigned unique identifiers (IDs) to mitigate any potential conflicts during the creation of new teams or games. Furthermore, both game names and team names must be unique within the system; hence, the application must incorporate a name-checking feature during the team and game creation processes.

Each round of the game will have a duration of one minute, with images rendered progressively during this time frame. After the initial 30 seconds, a brief interval will be allocated for other teams to submit their guesses, thereby enhancing the competitive aspect of the game. The real-time functionality is essential for sustaining an engaging experience for players, which necessitates a system capable of accommodating multiple clients concurrently connected to the same game session with minimal latency. To facilitate this, we will utilize WebSockets, which provide bidirectional communication between the client and server. This technology ensures that updates to the game state, including drawing progress and guesses, are promptly reflected on the screens of all connected players.

The 'Draw It or Lose It' game, accessible through a web platform, has the potential for significant user engagement. As such, your role in ensuring the system's scalability and security is crucial. The system architecture must be designed to handle fluctuating user loads, with the capability to scale in response to an increasing number of participants. Robust security measures, including secure communication protocols and user authentication mechanisms, are essential to prevent potential cheating or manipulation of game data.

## [Design Constraints](#_2et92p0)

The development of "Draw It or Lose It" as a web application presents several design constraints that must be addressed to ensure the system operates efficiently, securely, and responsively within a distributed environment.

A principal challenge lies in facilitating real-time communication among players. Given that the game involves multiple teams making guesses based on progressively rendered images, low-latency communication is essential to provide a seamless and synchronized experience. To achieve this, WebSockets will be utilized to establish persistent, full-duplex communication between the client and the server. This technology will enable real-time updates of both the drawing and guessing processes, ensuring that all participants are engaging with the most current version of the game state without any delays. However, the implementation of this feature necessitates careful consideration of network performance, as issues such as elevated latency or connection interruptions can adversely affect the user experience.

Another critical design constraint includes the requirement for unique identifiers for each game, team, and player. This is vital for preserving the integrity of the game state and ensuring that the correct associations are made between actions and their corresponding teams or players. The system must efficiently generate and manage these unique identifiers through the use of technologies such as universally unique identifiers (UUIDs) or a comparable approach. Additionally, unique naming for games and teams must be enforced within the system, thus necessitating a mechanism to verify name availability and prevent duplicates during the creation process.

Scalability also represents a significant consideration. The application must support multiple concurrent game instances, each comprising its teams and players. As user participation increases, the system must be capable of horizontal scaling to accommodate this increased demand. Cloud services such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud will be employed to provide the requisite infrastructure for scaling the application and managing the associated load. This will ensure sustained performance, even as the number of concurrent users rises.

Security and data integrity are of utmost importance. Given the real-time nature of the game, the system must preclude any manipulation of game data or attempts to cheat. Secure communications will be established through the use of HTTPS to encrypt interactions between clients and the server, while robust user authentication methods will be implemented to restrict access. Furthermore, the server will be designated as the authoritative source of game data to prevent unauthorized alterations or guesses by players.

These design constraints will significantly shape the system's overall architecture and developmental approach. The application will be designed with a focus on scalability, performance, and security, thereby ensuring that the web-based "Draw It or Lose It" game remains an engaging, reliable, and secure experience for all participants.

## [System Architecture View](#_ilbxbyevv6b6)

This section addresses critical aspects of the system architecture, which, while not mandatory for the current design, are nonetheless essential for understanding the overall framework. The architecture comprises multiple components, each fulfilling a specific function within the application. It adopts a tiered approach, incorporating distinct layers for the front-end, back-end, and real-time communication.

The front-end component, constructed with React, will manage user interactions and present game data, including the progressive drawing and current game state. The back-end, developed utilizing Node.js, will oversee game logic, user authentication, and the persistence of game states. Furthermore, WebSockets will facilitate real-time communication between clients and the server.

Data pertinent to the game, such as user accounts, teams, and game states, will be housed within a database. To efficiently manage both structured data, such as team and player details, and unstructured data, such as game states, a hybrid approach utilizing both relational and NoSQL databases may be employed. Additionally, a caching layer, such as Redis, will be used to store frequently accessed game state data, ensuring low latency access during real-time interactions.

The entire system is designed for deployment on a cloud platform, which will provide the necessary flexibility and scalability to accommodate varying user demands. This cloud environment will enable horizontal scaling, thus allowing the game to manage an increasing number of concurrent users effectively. The architecture will also incorporate load balancing and failover mechanisms, thereby ensuring high availability and reliability and giving you peace of mind about the system's performance.

In summary, the overarching system design is meticulously crafted to facilitate efficient communication, data management, and security. This design ensures a seamless and engaging experience for players while fulfilling the business requirements established by The Gaming Room.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram delineates a well-structured object-oriented design for a game management system. Central to this design is the GameService class, which is implemented as a Singleton. This implementation guarantees that a singular instance is responsible for overseeing the game's state, Gameoncept validated by the SingletonTester class.

The design adheres to the principles of inheritance, with the Game, Team, and Player classes deriving from the Entity class. This inheritance allows these classes to share common attributes, such as identification and name. Moreover, these classes exemplify composition through "1-to-many" relationships: the GameService manages multiple Game objects, each Game compasses multiple Team objects, and each Team comprises multiple Player objects.

Such a structure facilitates the management of interrelated entities and enhances encapsulation. Each class effectively conceals its internal data (including collections of teams or players) while providing methods for interaction with that data. The adoption of the Singleton pattern for GameService assures a singular access point to the game, which is essential for maintaining consistency and managing resources efficiently.

In summary, the diagram represents a clean and modular design that supports scalability and maintainability while effectively fulfilling the software's requirements.

**"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** | Unix-based OS with built-in security, stability, and compatibility with open-source tools.  Good for development environments and smaller-scale deployments.   High hardware costs, limited server configuration options, not ideal for large-scale production hosting. | Open-source, highly customizable, and optimized for performance.  Cost-effective, scalable, widely used in server environments, strong security and stability.  Requires Linux system administration expertise, some software may require extra configuration. | User-friendly interface, ideal for applications built with Microsoft technologies.   Broad compatibility with enterprise software, solid integration with Microsoft products.  Higher resource consumption, licensing costs, less flexibility and customizability compared to Linux. | Typically used for accessing applications, not for hosting.  Accessible from anywhere, offers mobile-first web development.  Lack of server-side capabilities, limited processing power, not suitable for hosting web applications. |
| **Client Side** | Requires macOS expertise and development tools like Xcode, Swift, and Objective-C for native applications.  Development tools are free, but hardware can be expensive.   Development can be time-consuming for cross-platform compatibility.  Need familiarity with macOS and its ecosystem, including application distribution via the Mac App Store. | Linux is open-source, and development is usually done using web technologies or cross-platform tools like Electron.  Free, but some software tools may require licenses or subscriptions.  Linux has a more niche user base, so additional time may be needed for testing and ensuring compatibility.  Requires developers to have familiarity with Linux-based development environments, though many tools are cross-platform. | Ideal for desktop applications, especially if using Microsoft technologies.  Free development tools (Visual Studio), but hardware costs can be higher for premium devices.  Windows offers a rich development ecosystem but may require additional effort for cross-platform support.  Requires knowledge of Windows development tools and technologies. | Development often uses mobile-specific SDKs (iOS: Swift, Android: Kotlin/Java), or cross-platform tools like React Native.  Requires purchasing hardware (iOS devices, Android devices) and potentially platform-specific licenses.  Mobile development typically involves separate codebases for iOS and Android, unless using cross-platform frameworks.  Developers need expertise in mobile SDKs, and platform-specific guidelines for user experience. |
| **Development Tools** | Xcode, Swift, Objective-C, and third-party tools like Visual Studio Code.  Typically uses web technologies (HTML, CSS, JavaScript) via tools like Sublime Text, VSCode, or Terminal.  Frameworks: Cocoa, Cocoa Touch, and web frameworks (e.g., React, Angular, Vue.js) | Supports a wide range of languages (Python, PHP, Ruby, Node.js, Java, C++).  Development tools: VS Code, Sublime Text, Atom, and command-line tools.   LAMP (Linux, Apache, MySQL, PHP/Python), Node.js, and Docker for containerization. | Languages & Tools: Visual Studio, C#, .NET, ASP.NET, PowerShell, and other Microsoft tools.  Web development: ASP.NET, IIS, C# for back-end, JavaScript/HTML5 for front-end.  Frameworks: .NET Core, WinForms, WPF for desktop apps. | Languages & Tools: Xcode (Swift, Objective-C for iOS), Android Studio (Kotlin, Java for Android), React Native, Flutter for cross-platform.  Frameworks: UIKit (iOS), Jetpack (Android), Xamarin, Flutter, or React Native for cross-platform development.  Development Tools: Mobile-specific IDEs (Xcode, Android Studio) and cloud-based testing platforms (e.g., Firebase, TestFlight). |

## 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" across various computing environments, Windows would be the ideal primary operating platform. With a dominant market share in gaming, Windows is the most widely used operating system among gamers. It provides robust support for high-performance applications, including graphics-intensive games, and offers a wealth of development tools and libraries, such as DirectX, tailored explicitly for game development. Furthermore, Windows supports a wide range of hardware configurations, which is essential for reaching a broad audience. This flexibility, combined with its extensive user base, makes Windows the most suitable option for effectively expanding the game to multiple environments.

1. **Operating Systems Architectures**:

Windows operates on a hybrid kernel architecture, combining elements of monolithic and microkernels. This design efficiently manages system resources while maintaining a balance of performance and security. By using multiple layers—user mode, kernel mode, and hardware abstraction—Windows isolates application processes from critical system operations. It supports multitasking, memory protection, and advanced process management, ensuring smooth performance for applications like "Draw It or Lose It" across various hardware. Additionally, extensive support for APIs such as Win32 and DirectX enhances developers' access to low-level resources, particularly for graphics-intensive applications.

1. **Storage Management**:

For Windows, a suitable storage management system would be a cloud storage solution such as Amazon S3 or Microsoft Azure Blob Storage. These platforms offer scalable, reliable, and redundant storage capable of handling large volumes of user-generated content, including game assets, player data, and real-time updates. Both Amazon S3 and Azure Blob Storage ensure high availability, automatic backup, and seamless integration with Windows-based environments, making them ideal for cloud-based data storage. Additionally, Windows supports file systems like NTFS for local storage, which effectively manages file storage, access control, and data integrity on the local machine.

1. **Memory Management**:

On Windows, memory management is carried out through a combination of virtual memory, paging, and segmentation to efficiently allocate and manage system resources for applications like "Draw It or Lose It." The system uses dynamic memory allocation, allowing programs to request memory as needed. The Windows Memory Manager ensures that memory is allocated and deallocated correctly, preventing memory leaks. It also utilizes paging to swap data between RAM and disk storage, enabling applications to run efficiently even when physical memory is limited. Furthermore, Windows supports automatic memory protection, isolating each application's memory space to prevent one program from interfering with the memory of another, ensuring stable performance for the game.

1. **Distributed Systems and Networks**:

To enable "Draw It or Lose It" to communicate across different platforms, the game can utilize cloud-based servers along with WebSocket or RESTful APIs for real-time data synchronization between devices. The distributed system should incorporate load balancing to manage traffic and ensure high availability effectively. Additionally, it should offer an offline mode to address network outages. In the event of connectivity issues, implementing local data caching and retry mechanisms will help maintain functionality until the network connection is restored.

1. **Security**:

Windows provides robust built-in security features to safeguard user information across different platforms. BitLocker offers full disk encryption, while Windows Defender delivers real-time malware protection. Communication between devices can be secured using SSL/TLS encryption to ensure that data is transmitted safely over the network. Additionally, user authentication should be managed with secure protocols such as OAuth 2.0 or JWT. It is also essential to encrypt sensitive data both at rest and in transit, ensuring strong protection against unauthorized access.