

# 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 | 08/25/2024 | Corderro Artz | Expanded and enhanced the Recommendations section to better align with the application requirements. |

**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 has tasked Creative Technology Solutions (CTS) with expanding their existing Android game, *Draw It or Lose It*, into a web-based, multi-platform application. The objective is to develop a solution that allows this game to be played on various platforms, including web, mobile, and desktop environments, while maintaining a seamless gaming experience across all devices.

The expansion of the game to a web-based platform introduces several technical challenges, including the need for a scalable and distributed system that can handle multiple players and teams simultaneously. The core requirements include ensuring the uniqueness of game and team names, managing multiple teams and players within a single game instance, and maintaining a single instance of the game in memory at any given time.

To meet these requirements, we propose a solution that leverages a web-based distributed architecture, ensuring scalability, reliability, and ease of deployment across different platforms. By implementing object-oriented design principles and using modern web technologies, we aim to deliver a robust and flexible gaming platform that meets the client's needs and provides a solid foundation for future enhancements.

## Requirements

The Gaming Room's primary business and technical requirements for the Draw It or Lose It game application are as follows:

1. ***Multi-Platform Support****: The game must be accessible on web browsers, mobile devices, and desktop environments.*
2. ***Scalability****: The system must support multiple concurrent games, each with multiple teams and players.*
3. ***Uniqueness****: Game and team names must be unique, preventing duplicate entries.*
4. ***Singleton Game Instance****: Only one instance of the game should exist in memory at any given time, ensuring consistency and preventing conflicts.*
5. ***User Experience****: The game should provide a seamless and responsive user experience across all platforms.*

## [Design Constraints](#_2et92p0)

Developing a web-based distributed game application introduces several design constraints that must be carefully managed to achieve the desired outcome. The key constraints include:

1. ***Platform Compatibility****:* The game must be designed to run seamlessly on web, mobile, and desktop platforms. This requires careful consideration of platform-specific limitations, such as screen sizes, input methods, and performance characteristics.
2. ***Scalability and Load Management****:* The application must be able to handle varying loads, from single-player games to multiple concurrent games with numerous teams and players. This necessitates a distributed architecture that can scale horizontally to accommodate increased demand.
3. ***Data Consistency and Synchronization****:* Ensuring that game and team names are unique across all instances requires a centralized mechanism for name validation and conflict resolution. Additionally, managing a single game instance in memory at any given time introduces challenges in data synchronization and state management across distributed systems.
4. ***Security****:* Given that the game will be accessible over the web, security is a significant concern. The application must protect user data, prevent unauthorized access, and ensure secure communication between clients and servers.

The implications of these design constraints are significant. They affect the choice of technologies, the architecture of the system, and the development process. For example, ensuring platform compatibility may require the use of responsive design techniques and cross-platform development frameworks.

Scalability concerns may lead to the adoption of microservices or cloud-based architectures. Data consistency challenges may necessitate the use of distributed databases or consensus algorithms. Finally, security requirements may influence the choice of encryption protocols and authentication mechanisms.

## [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 provided for the *Draw It or Lose It* game application represents the domain model, which is essential for understanding the relationships between the various entities within the system.

**"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.**

**Class Descriptions and Relationships:**

1. ProgramDriver: This class serves as the entry point for the application. It interacts with the SingletonTester class to initiate the game instance.
2. SingletonTester: This class is responsible for testing the singleton pattern implementation, ensuring that only one instance of the game is created and managed in memory.
3. GameService: This class functions as the service layer, managing the core game logic. It maintains a list of games, tracks the next available IDs for games, players, and teams, and provides methods to add and retrieve games by ID or name.
4. Game: The Game class represents a single game instance. It contains a list of teams and provides methods to add teams, and convert the game object to a string representation.
5. Team: The Team class represents a team within the game. It manages a list of players and provides methods to add players and convert the team object to a string representation.
6. Player: The Player class represents an individual player within a team. It contains methods to retrieve player information and convert the player object to a string representation.
7. Entity: This is a base class from which Game, Team, and Player inherit. It encapsulates common attributes such as id and name, along with methods for retrieving these attributes and converting the object to a string representation.

## Object-Oriented Programming Principles:

## Inheritance: The Entity class serves as a base class for Game, Team, and Player, demonstrating the principle of inheritance, which promotes code reuse and simplifies the management of common attributes and behaviors.

## Encapsulation: Each class encapsulates its data and provides methods for interacting with that data, ensuring that the internal state of objects is protected and only accessible through well-defined interfaces.

## Singleton Pattern: The GameService class implements the singleton pattern, ensuring that only one instance of the game service exists. This is crucial for maintaining a single game instance in memory and preventing conflicts.

## [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** | Mac OS offers stability and security but is less common for large-scale web hosting. Hardware and hosting options are more limited compared to Linux. Requires Xcode and apple hardware. | Linux is the leading choice for hosting due to its open-source nature, scalability, speed, and cost-effectiveness. It has extensive community support and a near unlimited amount of free and accessible resources. | Windows Server integrates well with Microsoft products and is user-friendly but has higher licensing costs and less flexibility than Linux. | Mobile devices are used primarily as clients, requiring responsive design and native app development, not server-side hosting. Though it possible, it is highly impractical. |
| **Client Side** | Like Server Side, Development on Mac requires Xcode and Apple hardware which can be costly. Though, it provides a stable and secure environment for users. | Linux is favored by developers, offering open-source tools and customizability. Compatibility across distributions can be complex but not uncommon. A great choice for client side development. | Windows is essential for client-side support due to its large user base. Development is traditionally done with Visual Studio, though licensing can be expensive. VSCode, JetBrains product line and other alternatives are now available and more cost effective. | Mobile development requires specific environments (Xcode for iOS, Android Studio for Android). Cross-platform compatibility is key, though it increases complexity. |
| **Development Tools** | Xcode, which supports Swift and Objective-C is well-integrated but limited to Apple platforms. | Linux offers a wide range of open-source tools and languages. It’s ideal for server-side development with strong support for automation and containerization. Multiple IDE’s and other tools can be found within Linux. | Windows development is centered around Visual Studio, which supports many languages and integrates with Azure. It’s powerful but comes with licensing costs. VSCode, JetBrains product line and other alternatives are now available and more cost effective. | Mobile tools vary by platform (Xcode for iOS, Android Studio for Android). Cross-platform tools like Flutter and React Native help streamline development. |

## Evaluation - Expanded

## Server Side

## Mac: While macOS provides a Unix-based environment conducive to development, it is not typically used for server hosting due to limited enterprise support and higher costs associated with Apple hardware. This makes it less suitable for scalable web applications where cost-effectiveness and flexibility are paramount.

## Linux: Its open-source nature, extensive community support, and minimal licensing costs make Linux the preferred choice for server-side deployments. It offers numerous deployment methods, including cloud-based and on-premises solutions, making it highly adaptable for a wide range of applications.

## Windows: Windows Server’s integration with the Microsoft ecosystem makes it a strong contender for organizations already invested in Microsoft technologies. However, the higher licensing costs and potentially complex configurations can be a drawback, especially when compared to Linux.

## Mobile Devices: Mobile platforms are not typically used for server-side hosting; instead, the focus is on ensuring that mobile clients can effectively communicate with back-end services hosted on cloud platforms like AWS or Azure.

## Client Side

## Mac: Developing for macOS and iOS requires Apple-specific tools and hardware, which can increase development costs. However, using cross-platform frameworks can mitigate some of these challenges by allowing code reuse across iOS and Android.

## Linux: Linux’s flexibility and open-source nature make it a strong choice for client-side development, particularly for developers who prefer or require a customizable environment. Ensuring compatibility across different distributions and desktop environments can be challenging but manageable with proper testing.

## Windows: With its large user base, Windows is crucial for client-side development. Visual Studio remains a powerful tool, though licensing costs can be high. Cross-platform tools can help streamline development for mobile and web, ensuring consistent experiences across devices.

## Mobile Devices: Development for mobile platforms requires specialized tools (Xcode, Android Studio), but cross-platform frameworks like Flutter or React Native can simplify the process, allowing for a single codebase to support multiple platforms.

## Development Tools

## Mac: The reliance on Xcode and Apple’s ecosystem means that developers must be proficient in Apple’s tools and languages. This could require a dedicated team for iOS/macOS development, potentially increasing costs and complexity.

## Linux: The availability of open-source tools and the ability to customize the development environment make Linux an attractive option for many developers. The minimal licensing costs and the flexibility of Linux environments can help keep development costs low.

## Windows: Visual Studio is a comprehensive development environment, but its licensing costs can be prohibitive for some projects. The need for expertise in Microsoft’s toolchain could necessitate specialized teams, though the availability of free alternatives like VSCode can mitigate some costs.

## Mobile Devices: Cross-platform tools like Flutter and React Native reduce the need for expertise in multiple mobile development environments, allowing for a unified development process. This can reduce the number of developers needed and lower overall costs.

## Recommendations

## To ensure the successful expansion of the *Draw It or Lose It* game application into a multi-platform, web-based environment, we provide the following recommendations:

## 1. Operating Platform

## Recommendation: We recommend using a Linux-based server platform, such as Ubuntu Server or CentOS, hosted on a cloud infrastructure like Amazon Web Services (AWS) or Microsoft Azure.

## Explanation:

## Linux is the industry standard for web servers, offering unparalleled flexibility, scalability, and cost-effectiveness. It is open-source, meaning there are no licensing fees, and it supports a vast array of tools and frameworks that are essential for deploying scalable web applications.

## By leveraging cloud services like AWS or Azure, The Gaming Room can benefit from scalable infrastructure, ensuring that the game can handle traffic spikes and grow as the user base expands. These platforms also offer robust support for Linux environments, including pre-configured virtual machines, managed databases, and load balancing services.

## 2. Operating Systems Architectures

## Recommendation: Implement a microservices architecture on the Linux-based server platform.

## Explanation:

## A microservices architecture divides the application into small, independent services that communicate over a network. This approach allows each component (such as user authentication, game logic, and data storage) to be developed, deployed, and scaled independently.

## This architecture enhances resilience, as individual services can be updated or scaled without affecting the entire system. It also aligns well with cloud environments, which support automatic scaling and load balancing, ensuring that the application remains responsive under varying loads.

## 3. Storage Management

## Recommendation: Utilize a NoSQL database like MongoDB for managing game data, combined with cloud-based storage solutions for handling large static assets (e.g., game images, logs).

## Explanation:

## MongoDB is well-suited for managing the hierarchical and unstructured data typical of game applications, such as player profiles, game states, and team information. It provides flexibility in data modeling, horizontal scalability, and high availability, which are crucial for supporting the dynamic nature of *Draw It or Lose It*.

## For static assets, cloud storage solutions like Amazon S3 or Azure Blob Storage offer scalable, durable, and cost-effective storage options. These services provide easy integration with the application and ensure that assets are quickly accessible to users across different platforms.

## 4. Memory Management

## Recommendation: Leverage Linux's built-in memory management techniques and cloud-based automated management features to ensure efficient use of memory.

## Explanation:

## Linux employs advanced memory management techniques, including virtual memory, paging, and caching, to optimize the use of system memory. These features help ensure that the game remains performant, even under heavy load, by efficiently managing the allocation and deallocation of memory resources.

## In a cloud environment, automated memory management features (like auto-scaling and garbage collection) further enhance performance by dynamically adjusting resource allocation based on current demand, preventing memory leaks, and ensuring that the application maintains optimal performance over time.

## 5. Distributed Systems and Networks

## Recommendation: Implement RESTful APIs and WebSockets to enable communication between the various components of the distributed system.

## Explanation:

## RESTful APIs provide a standardized way for different parts of the application (such as the game server, client apps, and databases) to communicate over HTTP. This approach ensures compatibility across different platforms and devices, making it simpler to maintain and extend the application.

## WebSockets offer real-time, bidirectional communication, which is essential for the interactive nature of *Draw It or Lose It*. This allows the server to push updates to clients instantly, ensuring that all players see the same game state and can interact seamlessly.

## The cloud environment's distributed nature ensures high availability and fault tolerance. By distributing the application across multiple geographic regions, the system can continue to operate smoothly even in the event of localized outages or high traffic in a specific area.

## 6. Security

## Recommendation: Implement end-to-end encryption, secure authentication mechanisms, and regular security audits to protect user data.

## Explanation:

## End-to-end encryption ensures that all data transmitted between the client and server is protected from interception. Using TLS (Transport Layer Security) for all communications is essential to safeguard sensitive information, like player credentials and game data.

## Secure authentication mechanisms, such as OAuth 2.0 or JWT (JSON Web Tokens), should be employed to verify user identities and control access to the application. This approach ensures that only authorized users can access the game, and that their sessions are protected.

## Regular security audits and vulnerability assessments should be conducted to identify and mitigate potential security risks. This proactive approach to security will help protect against threats such as SQL injection, cross-site scripting (XSS), and distributed denial-of-service (DDoS) attacks.

## Cloud providers like AWS and Azure offer comprehensive security tools, such as AWS Shield for DDoS protection and Azure Security Center for continuous monitoring, which should be integrated into the overall security strategy.