

Draw It or Lose It

# **CS230 Project Software Design Document**

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 |
| --- | --- | --- | --- |
| 3.0 | 07/19/25 | Brooks Maerder | Final draft of the software design document including executive summary, requirements, design constraints, domain model analysis, evaluation, and platform 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)

Creative Technology Solutions (CTS) has been contracted by The Gaming Room to assist in the development of a web version of their current Android exclusive game, Draw It or Lose It. The client’s objective is to expand the platform reach of the game to support web access, allowing cross-platform play and enhance user accessibility. The game revolves around timed guessing challenges based on visual clues, with teams competing over several rounds. To support this expansion, a robust, scalable, and secure solution is needed that accommodates team management, unique player and game identification, and play time constraints for each round. This document outlines the proposed software design, domain model interpretation, and development considerations to meet these business goals efficiently and effectively.

## Requirements

#### *****Business Requirements:*****

* The game must be accessible across multiple platforms (desktop and mobile) via a web browser to increase audience reach.
* The user experience must be consistent with the original Android app to maintain brand familiarity and player retention.

#### ****Technical Requirements:****

* The web application must support real time gameplay, including countdown timers, image rendering sequences, and synchronized updates across teams.
* All game, team, and player names must be unique and validated in real-time at the point of creation.
* Only one active instance of each game session can be held in memory at a time, using unique game identifiers.
* The system must be designed for scalability and availability, capable of supporting many concurrent users.
* The software must be secure, protecting user data and ensuring authorized access to game sessions.
* Backend infrastructure must support team management, session control, and efficient memory use for active game logic.

## [Design Constraints](#_2et92p0)

* **Platform Independence:** The application must operate across multiple platforms and browsers, which constrains development to using cross-platform technologies such as HTML5, JavaScript, or CSS.
* **Single Instance Enforcement:** Only one instance of the game may exist in memory at a time, requiring a centralized management system with unique identifiers for games, teams, and players. This suggests the use of a singleton pattern to be implemented in the web game.
* **Real-Time Interaction:** Timed rounds and simultaneous team interactions demand real-time communication, implying the use of technologies such as WebSockets for live updates.
* **Data Integrity and Uniqueness:** Names for games and teams must be unique, which requires real-time validation and conflict resolution mechanisms both client and server side.
* **Scalability:** As player count may vary, the system must scale efficiently in a distributed environment, potentially via cloud infrastructure.

## [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 provided UML class diagram represents the domain model for Draw It or Lose It. At the core is an abstract Entity class, which encapsulates shared attributes like unique identifiers and possibly timestamps or audit fields. Inheritance from this base class ensures reusability and enforces a consistent structure across derived classes.

Key components include:

* **Game**: The central object, representing a session of the game. It is linked to one or more Team objects and maintains the current state of the game, such as active round and time remaining.
* **Team**: A composite class containing multiple Player objects. Teams have unique names and are associated with a specific game.
* **Player**: An individual participant, associated with one team only.

Object-Oriented Programming (OOP) principles are evident in this structure:

* **Encapsulation**: Each class encapsulates its own data and behavior.
* **Inheritance**: Common features are centralized in the Entity base class.
* **Association**: Classes are linked appropriately to reflect game logic (e.g. one-to-many between Player and Team).
* **Uniqueness Constraints**: Enforced through validation logic embedded in the design, possibly using static maps or database-level constraints.

"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 is Unix-based and supports many modern web technologies. However, it's less commonly used in enterprise hosting environments, which may limit third-party hosting options and scalability. | Linux is the preferred platform for hosting due to its high performance, open-source nature, and scalability. It supports essential technologies and offers robust community and commercial support. | Windows Server can host web apps but may require more resources and licenses. It supports ASP.NET well, but for cross-platform apps, it is generally less cost-effective than Linux. | Mobile devices are not suitable for hosting due to limited processing power and stability. Hosting should remain on server-class machines while mobile clients consume data. |
| **Client Side** | Mac clients can access the game easily via modern browsers. Development requires attention to UI consistency due to different input styles and screen sizes. | Linux desktops are less common among end users but support all major browsers. Development must ensure proper rendering in Firefox, Chromium, etc. | Windows is the most widely used desktop OS and a major target for client support. Ensuring compatibility with Edge and Chrome is essential. | Mobile devices (iOS/Android) are key targets. Development must ensure responsive design and possibly progressive web apps (PWAs) to mimic native app behavior. |
| **Development Tools** | macOS supports tools like VS Code, Node.js, and Git. Xcode may be used for iOS-specific builds if mobile support is expanded. | Linux is ideal for backend development using Node.js, Docker, and databases like MySQL/PostgreSQL. Command-line tools are powerful and widely supported. | Windows supports IDEs like Visual Studio and tools like Git Bash. Windows Subsystem for Linux (WSL) can also enable Linux-like development environments. | Mobile app dev requires tools like Android Studio or Xcode. For web support, JavaScript frameworks like React Native or Flutter for hybrid solutions are viable. |

## 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**: CTS should recommend Linux as the primary server side operating platform. It is highly scalable, secure, and cost-effective for hosting web applications. On the client side, support should focus on web browsers across Windows, Mac, and mobile platforms (iOS/Android).
2. **Operating Systems Architectures**: The game should use a distributed client-server architecture with clearly defined tiers. The presentation layer will run in the user’s web browser on Windows, macOS, Linux, iOS, or Android, built using HTML5, CSS, and JavaScript with a responsive front-end framework such as React. This layer is responsible for rendering the interface, handling user input, and communicating with the server via secure HTTPS and WebSocket connections. The application layer will run on a Linux based server hosting a Node.js or similar runtime environment to manage game logic, user sessions, timers, input validation, and enforcement of unique players and teams. The data layer will use a cloud hosted MySQL database to store persistent data such as accounts, teams, and game history, with uniqueness constraints and rapid retrieval supported by a caching service like Redis. Real time gameplay will be enabled through WebSockets for synchronous updates, while asynchronous operations such as account creation or history retrieval will be handled via RESTful APIs. To support scalability and availability, the architecture will be designed with additional application servers behind a load balancer, ensuring high performance during peak usage and protection against server outages.
3. **Storage Management**: In evaluating file system options across potential operating platforms, Linux is optimized for high performance web hosting, large file handling, and quick metadata access which is ideal for rapid retrieval of game state and user data. Windows servers typically offer strong access control features but may introduce higher overhead in certain high concurrency web applications. MacOS is well suited for SSDs and encryption but is less common in large scale hosting environments. Given the distributed nature of Draw It or Lose It and the need for scalability, the Linux file system provides the best balance of reliability, performance, and compatibility with cloud based MySQL storage.
4. **Memory Management**: Efficient memory usage is critical for supporting real time gameplay and ensuring system stability under high loads. The recommended Linux based server platform will implement multiple memory management techniques tailored to Draw It or Lose It. Active game states will be stored in volatile memory using a high speed caching service such as Redis allowing rapid access times for in game timers, player turns, and score tracking. Expiration policies will automatically free memory when a game ends or becomes inactive, enforcing the single game instance rule and preventing stale data retention. The application will use stateless RESTful APIs for non real time operations, reducing persistent memory demands and allowing horizontal scaling without session conflicts. WebSocket connections will maintain only minimal state data on the server, offloading unnecessary information to the database or cache as needed. Garbage collection in the Node.js runtime will ensure unused objects are reclaimed quickly, and memory profiling tools will be used to detect and resolve potential leaks. This layered approach ensures optimal performance, predictable memory usage, and stability even during a high volume of gameplay sessions.
5. **Distributed Systems and Networks**: The game will operate as a distributed application where clients connect over the internet to a central web server. Real time gameplay interaction will be enabled using WebSockets to allow synchronous updates across players. The system will need to handle interruptions gracefully, ensuring players can reconnect and resume if connectivity is lost.
6. **Security**: User data must be protected through a multi layered approach that addresses both platform specific security features and secure communication between devices. All data in transit will be encrypted to prevent interception, and sensitive information such as passwords will be hashed before storage in the cloud-hosted MySQL database. On the Linux server, security will include firewall rules, intrusion detection systems, regular patching, and principle of least privilege user access. Platform specific browser security features such as sandboxing will be leveraged to protect client devices. For distributed gameplay, WebSocket connections will be secured with authentication tokens to ensure only authorized players can send or receive updates. Resource sharing policies will be strictly configured to prevent unauthorized requests, and input validation will be implemented both client and server side to protect against injection attacks. Finally, backup and disaster recovery plans will ensure data integrity in case of system failure or malicious activity, with regular testing to verify recovery processes.