

Draw it or lose it

# **CS 230 Project Software**

Version 3.0

## 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 | 01/25/2025 | Hamad Alghaithi | First iteration |
| 2.0 | 02/08/2025 | Hamad Alghaithi | Added multi-platform evaluation and deployment recommendations |
| 3.0 | 02/22/2025 | Hamad Alghaithi | Included system architecture analysis, storage and memory management, distributed systems, and security recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

This document presents the software design for "Draw It or Lose It," a web-based adaptation of the current Android application owned by The Gaming Room. The goal is to transition the game from a single-platform application to a multi-platform web-based system that allows players across various devices to participate in real-time. Our solution leverages modern web technologies to create an interactive, scalable, and easily maintainable gaming environment. Key features of the design include support for multiple teams and players, unique identifiers for games and teams, and real-time game state management to ensure a consistent and fair gaming experience for all participants.

## Requirements

*The Gaming Room has outlined the following business and technical requirements for the "Draw It or Lose It" web-based game:*

* **Multiple Teams & Players:** The game must allow for multiple teams, each consisting of several players.
* **Unique Names:** The game, team, and player names must be unique across the platform to prevent duplication and confusion during gameplay.
* **Single Instance of the Game:** Only one active instance of the game can exist at any given time, managed through unique game identifiers.
* **Game Rounds:** Each game consists of four rounds, each lasting one minute, with a steady stream of images presented as clues for players to guess the puzzle.
* **Platform Independence:** The game should work seamlessly across different platforms including mobile devices and desktop computers.
* **Real-time Functionality:** The game must be able to update the game state in real-time across all connected users, ensuring synchronized play.
* **Security:** The system must secure user data and protect personal information while preventing unauthorized access.

## [Design Constraints](#_2et92p0)

**Cross-Platform Compatibility:** The web-based nature of "Draw It or Lose It" requires it to function seamlessly across various platforms including desktops and mobile devices. This necessitates a responsive design that adapts to different screen sizes and resolutions.

**Scalability:** The system must handle a potentially large number of concurrent users without degradation of performance. This will impact the choice of server infrastructure, data handling, and client-server communication methods.

**State Management:** Managing the game state across distributed systems poses a challenge, particularly in ensuring that all players see the game updates simultaneously.

**Unique Identifiers:** Ensuring that game sessions, team names, and player identifiers are unique across the platform to prevent conflicts during game setup and play.

## [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 domain model for "Draw It or Lose It" is designed to efficiently manage game data and player interactions. The “Entity” class is the base for “Game”, “Team”, and “Player” classes, encapsulating common properties like ‘id’ and ‘name’ which are essential for database operations and client-server interactions. The “GameService” class acts as a singleton to ensure that only one instance manages the game logic, crucial for maintaining integrity and state consistency. The relationships between classes are managed through lists and IDs, facilitating easy tracking and updates of game components during runtime.

**"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.**

## 

## ؤ

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac can host web-based applications but is generally more costly due to its hardware and software ecosystem. It is rarely used in production environments as a server but is often favored for client applications, especially for iOS integration. | Linux is widely used for web servers due to its high performance, stability, and open-source nature, making it ideal for a cost-effective and scalable solution. It is the recommended server platform for the game application to minimize licensing costs. | Windows Server is commonly used in enterprises and integrates well with corporate infrastructure, though it requires higher licensing costs compared to Linux. Its stability and support for .NET applications make it suitable for certain server needs. | Not intended for server-side hosting; however, mobile devices interact with server-side components in real-time, demanding efficient, responsive server communication. |
| **Client Side** | Mac serves as an excellent development platform for client applications, especially for iOS, though it has a smaller market share among users. Browser-based games and web applications run smoothly on macOS. | Linux isn’t commonly used for consumer desktop applications, but it is solid for game development environments, and browser-based games perform well on it. | Windows is the most prevalent OS in the desktop market, making it essential to optimize the game for this platform. Windows-based games or browser-based games can perform optimally with dedicated configuration. | Mobile devices are crucial for reaching a broad user base, especially for casual games. The game design should be responsive and optimized for touch-screen compatibility, considering both iOS and Android devices. |
| **Development Tools** | Mac supports various development tools, including Xcode, IntelliJ, and Eclipse, as well as mobile development through Flutter and React Native. | Highly versatile with access to numerous IDEs and programming languages, Linux supports open-source tools and various frameworks essential for server and client application development. | Windows supports a wide range of development tools, including Visual Studio, Eclipse, and IntelliJ, compatible with most popular languages used for game development. | Cross-platform tools such as React Native and Flutter are recommended to ensure smooth user experience on both iOS and Android devices. |

Recommendations

Based on the evaluation, I recommend the following:

1. **Operating Platform:**
   * **Recommended Platform**: Linux should continue as the primary server platform for hosting. It is known for its reliability, scalability, and cost-effectiveness in high-demand environments.
   * **Rationale**: Linux’s stability and support for distributed architectures make it ideal for managing the load from a multi-platform game like "Draw It or Lose It," as it expands to support both mobile and desktop environments.
2. **Operating System Architectures:** A Linux-based architecture is recommended for the server environment, while a combination of mobile-friendly web interfaces will ensure cross-platform compatibility for both desktop and mobile clients.
   * **Description of Linux Architecture**: The Linux operating system features a modular kernel design with capabilities for multi-user access, advanced memory handling, and process isolation. These characteristics make Linux particularly effective for applications requiring resource sharing and inter-process communication.
   * **Benefits for Expansion**: Linux’s modular structure and robust networking capabilities allow for better system performance, efficient process management, and secure communication across various platforms, all of which are essential for a game requiring real-time updates and responsiveness.
3. **Storage Management:** Cloud-based storage solutions such as Amazon S3 or Google Cloud Storage are recommended for handling game data, providing scalability, redundancy, and ease of access.
   * **Recommended Storage Solution**: A combination of cloud-based storage, such as Amazon S3 or Google Cloud Storage, paired with a distributed database (e.g., MongoDB or PostgreSQL), is ideal.
   * **Rationale**: Cloud storage provides the scalability and redundancy necessary for supporting high-traffic applications with rapid data retrieval needs. Additionally, Linux file systems, like ext4 or ZFS, offer reliable local storage options for caching, backup, and log storage, enhancing system reliability and performance.
4. **Memory Management:** Implement memory management techniques, including object pooling and lazy initialization, to optimize memory usage during gameplay. For Linux servers, proper memory allocation and deallocation practices should be employed to avoid excessive memory usage.
   * **Memory Management Techniques in Linux**: Linux uses paging, segmentation, and swap space to manage memory effectively. This enables the operating system to handle large amounts of data, even under high user load, by balancing active and inactive memory.
   * **Application-Specific Needs**: For the high concurrency demands of "Draw It or Lose It," memory caching tools like Redis can be employed to reduce latency, ensuring a smooth player experience with faster response times during gameplay.
5. **Distributed Systems and Networks:** The use of RESTful APIs along with WebSocket connections will facilitate real-time interactions and updates among clients across platforms. A distributed microservices architecture will allow independent scalability and management of each system component.
   * **Communication in a Distributed Environment**: A microservices architecture using RESTful APIs and WebSockets is recommended for real-time data sharing and synchronization between server and client. This modular design allows the game to scale efficiently by enabling independent updates and maintenance for each game component.
   * **Network Dependencies and Fault Tolerance**: To handle network dependencies and potential outages, a distributed system should include load balancers and replication to maintain service continuity. Kubernetes can manage containerized microservices, distributing resources efficiently across nodes and ensuring fault tolerance.
   * **Cross-Platform Interaction**: REST APIs support seamless cross-platform communication, enabling consistent interaction between desktop and mobile users.
6. **Security:** To ensure security, SSL encryption should be used for all client-server communications. Implement OAuth for secure user authentication and hash sensitive data like passwords using bcrypt. Multi-factor authentication (MFA) can further secure access to user accounts.
   * **User Data Protection**: SSL/TLS encryption should secure data in transit, while data at rest should use strong encryption, such as AES-256. These measures ensure data privacy during client-server interactions.
   * **Authentication and Access Control**: Implement OAuth 2.0 for user authentication, with Multi-Factor Authentication (MFA) for additional security on sensitive operations. Access controls should enforce strict data access permissions to prevent unauthorized access.
   * **Platform-Specific Security**: Linux offers built-in security tools, including iptables for firewall configuration and SELinux for access control. Regular system patching will mitigate vulnerabilities. For mobile applications, App Transport Security (ATS) on iOS and Network Security Configuration on Android should be applied to ensure secure mobile interactions.