

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 01/26/25 | Tyler Miller | This revision includes the initial creation of key sections, including the Executive Summary, Requirements, Design Constraints, Domain Model, Evaluation of various operating platforms and mobile devices, and Recommendations for the preferred operating platform and architecture.  Top of Form |
| 2.0 | 2/9/2025 | Tyler Miller | Evaluated the characteristics, advantages, and weaknesses of Linux, Mac, Windows, and mobile platforms. |
| 3.0 | 2/23/2025 | Tyler Miller | Recommending operating platform, detailing the chosen operating system architectures, identifying a suitable storage management system, explaining the memory management techniques employed by the platform, outlining how distributed systems and networks can facilitate communication between various platforms, and addressing the security measures. |

**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 seeks to expand its Android-based game, Draw It or Lose It, into a web-based application accessible across multiple platforms. This expansion requires a scalable solution that ensures seamless gameplay, supports multiple teams with unique identifiers, and maintains centralized game state management. Creative Technology Solutions proposes a distributed web-based architecture that leverages a responsive frontend and a secure, scalable backend. By implementing object-oriented principles such as encapsulation, inheritance, and singleton patterns, the solution will efficiently meet the client’s requirements while ensuring scalability, reliability, and ease of maintenance. This document outlines the design approach, evaluation, and recommendations to guide the development process.

## Requirements

* **Game Structure:** Support for one or more teams, each with multiple players.
* **Uniqueness:** Unique identifiers for game, team, and player names.
* **Game State Management:** Only one instance of the game exists in memory at any given time.
* **Cross-Platform Compatibility:** Application accessible on web and adaptable to various platforms.
* **Performance and Scalability:** Efficient rendering and seamless user experience regardless of team size.

## [Design Constraints](#_2et92p0)

1. **Concurrency Management:**

* Handling simultaneous game sessions with multiple teams and players requires careful management of resources and thread safety.
* Implication: Use of design patterns like Singleton to ensure single-instance management for critical resources.

1. **Platform Independence:**

* The solution must work across different devices and operating systems.
* Implication: The backend will be developed using platform-agnostic technologies such as Java or Node.js, and the frontend will leverage responsive web technologies like HTML5, CSS3, and JavaScript frameworks.

1. **Network Reliability:**

* Distributed environments depend heavily on stable network connectivity.
* Implication: Implement fallback mechanisms and ensure synchronization through robust APIs.

1. **Data Security:**

* Protecting user data and ensuring secure communication between clients and the server is critical.
* Implication: Employ secure protocols such as HTTPS, and SSL/TLS and proper authentication mechanisms.

1. **Scalability:**

* The system must handle an increasing number of users and teams as the game grows in popularity.
* Implication: Use scalable architecture such as cloud-based infrastructure and load balancing.

## [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 represents the structure and relationships within the application. Below is a breakdown of key components:

1. **Entity Class:**

A base class holding common attributes (id, name) and methods like toString(). This promotes reusability and adheres to the principle of abstraction.

1. **GameService Class:**

Manages the collection of games, teams, and players using lists. Singleton design pattern ensures only one instance of this class exists, meeting the requirement of centralized game state management.

1. **Game Class:**

Represents a game instance. Maintains a list of teams and includes methods for adding teams and string representation. This demonstrates encapsulation by hiding team management details.

1. **Team Class:**

Represents a team within a game. Maintains a list of players and includes methods for adding players and string representation. Inheritance from Entity ensures consistent handling of shared attributes.

1. **Player Class:**

Represents individual players within a team. Inherits from Entity, promoting code reuse.

1. **ProgramDriver and SingletonTester Classes:**

Utility classes to initialize the game and test the Singleton implementation, respectively. These demonstrate the use of the <> relationship for testing and initialization purposes.

**Object-Oriented Principles Demonstrated:**

* **Abstraction:** Entity class abstracts shared attributes.
* **Encapsulation:** Methods like addGame and addTeam restrict direct access to underlying data structures.
* **Inheritance:** Game, Team, and Player classes inherit from the Entity class.
* **Singleton:** GameService class ensures centralized management of game instances.

**"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** | **Characteristics:** Stable and reliable for development.  **Advantages:** High-quality hardware and software integration. Secure and stable environment. Supports Docker and other containerization tools.  **Weaknesses:** Expensive hardware and limited server hosting use cases. Requires additional expertise for enterprise hosting setups. | **Characteristics:** Open-source, highly customizable, and efficient for hosting web applications.  **Advantages:** Cost-effective, scalable, and supports robust hosting technologies like Docker, Kubernetes, and Apache. Ideal for server environments with strong community support.  **Weaknesses:** Requires specialized knowledge for setup and maintenance. Less user-friendly compared to Mac or Windows. | **Characteristics:** User-friendly and widely adopted in corporate environments. Strong compatibility with enterprise solutions.  **Advantages:** Broad user base, extensive enterprise tools, and support. Easy to integrate with Microsoft products and services.  **Weaknesses:** Higher licensing costs and less flexibility compared to Linux. Vulnerable to security issues without proper maintenance. | **Characteristics:** Limited for hosting, but critical for accessibility. Ideal for front-end clients.  **Advantages:** High market penetration and mobile-friendly technologies like Progressive Web Apps (PWA). Allows real-time updates and interaction.  **Weaknesses:** Resource-constrained compared to traditional servers, making it unsuitable for backend hosting. Heavily dependent on network stability. |
| **Client Side** | **Characteristics:** Mac provides seamless user experience and has a loyal user base.  **Advantages:** High-quality graphics and performance. Fewer compatibility issues with macOS software.  **Weaknesses:** Higher costs for development and hardware. Requires expertise in macOS-specific tools (e.g., Xcode). | **Characteristics:** Linux is flexible and supports a wide variety of frameworks and programming languages. **Advantages:** Free to use, highly secure, and suitable for cross-platform development. Widely supported by developers. **Weaknesses:** Requires specialized skills for Linux-based client-side development. Compatibility testing is essential to ensure uniform performance across distributions. | **Characteristics:** Windows has a large user base and extensive development tools. Common in gaming and enterprise sectors.  **Advantages:** Highly compatible with existing systems and supports a wide range of software. Excellent documentation and community support.  **Weaknesses:** Licensing costs increase development expenses. More prone to malware and security issues without proper safeguards. | **Characteristics:** Mobile platforms (Android and iOS) are critical for end-user accessibility. Diverse device ecosystems require platform-specific development.  **Advantages:** Large user base, critical for engagement and usability. Tools like Android Studio and Xcode streamline mobile development.  **Weaknesses:** High maintenance cost for cross-platform development. Testing is required across numerous device models and operating system versions. |
| **Development Tools** | **Characteristics:** Requires tools like Xcode, IntelliJ IDEA, and VS Code. macOS provides an optimized environment for these.  **Advantages:** Excellent developer support for building high-performance applications.  **Weaknesses:** Limited to Mac hardware, increasing costs for teams. | **Characteristics:** Supports open-source tools like Eclipse, IntelliJ IDEA, and VS Code. Works with multiple programming languages.  **Advantages:** Cost-effective tools with wide adoption among developers.  **Weaknesses:** Developers must be familiar with Linux-specific configurations and tools. | **Characteristics:** Compatible with tools like Visual Studio, IntelliJ IDEA, and VS Code.  **Advantages:** Seamless integration with Microsoft tools. Supports a broad range of languages and frameworks.  **Weaknesses:** Requires licensing for development tools. Security updates and patches must be consistently applied. | **Characteristics:** Mobile platforms leverage Android Studio (for Android) and Xcode (for iOS). Cross-platform tools like Flutter or React Native are also used.  **Advantages:** Streamlines development for mobile-friendly applications. Provides platform-specific features.  **Weaknesses:** Higher complexity and cost for supporting multiple mobile platforms. Performance can vary across devices. |

## 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**: Linux is the ideal operating platform for hosting the web application due to its cost-effectiveness, scalability, and compatibility with modern development tools like Docker and Kubernetes. Linux's open-source nature ensures continuous updates and strong community support, making it a reliable and flexible choice for managing server-side resources.
2. **Operating Systems Architectures**: A microservices architecture is recommended, hosted on Linux. This architecture divides the application into modular components, each handling a specific aspect of the system, such as user authentication, game state management, and data storage. Microservices allow for independent updates and scaling of specific components, ensuring flexibility and reducing downtime during maintenance or feature deployment.
3. **Storage Management**: Cloud-based storage solutions like AWS S3 or Google Cloud Storage are recommended for managing game data, stock images, and user information. These platforms offer high availability, scalability, and robust data backup and recovery options. Cloud storage also supports integration with other services, such as CDN for faster content delivery, ensuring a smooth user experience even during peak usage.
4. **Memory Management**: Implementing in-memory caching systems like Redis or Memcached is recommended for efficient memory management. These systems store frequently accessed data in memory, such as game session data and player states, reducing database load and improving application response times. Caching also ensures real-time updates for users, which is critical for a fast-paced game like "Draw It or Lose It."
5. **Distributed Systems and Networks**: To ensure easy communication between various platforms, a distributed system with RESTful APIs or GraphQL is recommended for backend communication. WebSockets should be used for real-time updates, such as player moves and game state changes. Dependencies between components, such as backend servers, databases, and client devices, will be managed through load balancers and redundancy setups to handle connectivity issues and outages. For instance, use auto-scaling groups and load balancers to ensure uninterrupted service during peak traffic. Implement failover mechanisms and data replication to minimize downtime. Real-time communication between devices will be managed using WebSockets for instant updates, while periodic data snapshots ensure state recovery in case of interruptions.
6. **Security**: Key security measures include:

**Data Encryption:** Use HTTPS for secure data transmission and encrypt sensitive data, such as passwords and personal information, using algorithms like AES or RSA.

**Authentication:** Implement OAuth 2.0 or token-based authentication to ensure secure user login and session management.

**Platform Security:** Linux offers robust security features like SELinux and AppArmor to control application behavior and prevent unauthorized access.

**Data Protection:** Regular security audits, firewalls, and intrusion detection systems will safeguard the system from potential threats. Additionally, encrypted backups will protect user data during storage and recovery processes.