

Draw It or Lose It

# **CS 230 Project Software Design Template**

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 |
| --- | --- | --- | --- |
| 3.0 | 04/014/2025 | Aaron Befus | Implemented Recommendations section. |

## [Executive Summary](#_sbfa50wo7nsh)

*The Gaming Room is preparing to expand its popular mobile game Draw It or Lose It into a web-based version that supports multiple platforms. To support this effort, Creative Technology Solutions has designed a foundational software structure that will serve as the core of the new application. This structure enables multiple teams and players to participate in a single game session, ensures unique naming to avoid user confusion, and delivers consistent gameplay across all platforms.*

*The proposed solution is built for scalability, stability, and long-term maintainability. It allows The Gaming Room to manage games efficiently in memory, preventing duplication and ensuring each game, team, and player is uniquely identifiable. This back-end foundation sets the stage for future development, including the integration of user interfaces, gameplay features, and cross-platform functionality. With this architecture in place, The Gaming Room is well-positioned to move forward with full development of a modern, multi-platform gaming experience.*

## Requirements

1. *Platform Expansion* – The game must operate consistently across all platforms, ensuring all users experience the same version of the game in real time.
2. *Scalability* – The solution must be scalable to accommodate future growth, including additional game features, platform compatibility, and a growing user base.
3. *Data Integrity* – The system must maintain data integrity during gameplay, ensuring that only one version of a game is active at any time to avoid duplication or user conflicts.
4. *Efficient System* – The application must support real-time management of game sessions, teams, and players in an efficient and centralized way.
5. *Security* - The system must protect user input, session data, and game state from tampering or unauthorized access.

## [Design Constraints](#_2et92p0)

1. *Distributed Environment –* The application must function in a distributed environment, requiring careful handling of real-time synchronization and state sharing across users and platforms.
2. *Scalability* – The design must support future growth and modular expansion without requiring a full redesign.
3. *Cross-Platform Compatibility –* The application must remain flexible and portable, limiting reliance on platform-specific tools.
4. *Memory Management* – Only one instance of each game may exist in memory at a time, requiring precise control over game creation and access.
5. *Real-Time Performance* – The application must respond quickly to user actions and manage time-sensitive gameplay without lag.

## [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)

At the core of this system is the Entity class, which provides a common structure for its subclasses: Game, Team, and Player. This use of inheritance, a core object-oriented programming (OOP) principle, promotes code reuse and consistency by allowing these classes to share common attributes such as id and name, and common behaviors such as getId(), getName(), and toString().

The GameService class is responsible for managing all active games. It uses a singleton design pattern, to ensure that only one instance of itself exists at a time, which is important for maintaining a centralized and consistent game state in memory. It holds a list of all Game objects and includes methods to add and retrieve games and teams, while enforcing name uniqueness.

Each Game contains a list of Team objects, and each Team contains a list of Player objects. This demonstrates a composition relationship, where higher-level objects are composed of collections of lower-level objects, forming a clear hierarchy: GameService → Game → Team → Player.

This hierarchy supports the need for efficient management of real-time game sessions by allowing for structured traversal and manipulation of objects at different levels. For instance, the application can quickly locate a game, then identify and manipulate its teams and players.

The model also demonstrates encapsulation by keeping class attributes private and exposing behavior through public methods. This improves maintainability and protects the integrity of game data.

**"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** | While Mac has built-in support for Apache and other server tools, it lacks a modern, dedicated server edition. macOS can also only be run from Apple hardware, which is significantly more expensive than typical server-grade alternatives. This hardware requirement introduces a major cost consideration. Combined with limited scalability and support in cloud environments, macOS is not a practical or cost-effective choice. | Linux offers many distributions that are optimized for web hosting and are supported by major cloud providers (e.g. Ubuntu Server, CentOS, and Debian). These environments are stable, secure, and highly customizable, making them ideal for hosting scalable applications like Draw It or Lose It. Linux is open source and free to use, so there are no licensing fees involved. s | Windows Server is a well-established platform for web-based application hosting and is supported by all major cloud providers. It offers an intuitive management interface and integration with Microsoft development tools. However, unlike Linux, Windows Server requires paid licenses, either through a subscription model or per-instance pricing in the cloud/ | iOS and Android are not designed to host server-based applications and do not offer native server deployment capabilities. Devices running these operating systems function strictly as clients, accessing services hosted on other platforms like Linux or Windows. Hence, there are no licensing costs involved. |
| **Client Side** | The primary concern to support Mac is ensuring that the application runs smoothly in web browsers commonly used on macOS, particularly Safari, Chrome, and Firefox. Development costs and time for macOS are moderate, as most tools used to build and test web applications – such as Visual Studio Code and Chrome DevTools – are fully supported on the platform. | Supporting Linux as a client involves ensuring that the web application functions correctly across a variety of browsers and desktop environments. No platform-specific development is needed for Linux. However, because distributions vary widely, consistent performance requires thorough cross-browser testing, adherence to web standards, and testing of adaptation to differing window managers. | Windows remains the most widely used desktop operating system, so ensuring compatibility with its popular browsers – Chrome, Edge, and Firefox – is critical. Development does not require platform-specific adaptations for Windows. Development time and cost are moderate, with ample support from tools like Visual Studio Code, browser dev tools, and automated testing frameworks | To support both iOS and Android platforms, the application must be developed using responsive web design principles that adapt the user interface to various screen sizes, orientations, and touch interactions. Ensuring compatibility with Safari on iOS and Chrome on Android is essential. Development time may increase due to the need for testing across devices. |
| **Development Tools** | macOS supports a wide range of modern development tools relevant to this project (e.g. Xcode). Developers working on Mac can also utilize terminal-based tools, Git for version control, and Docker for containerization and testing. Most of these tools are open source or offer free community editions, so licensing costs are minimal. macOS supports full-stack development, and there is no need for a separate team if the developers are trained in cross-platform web technologies. | Linux provides a robust and flexible environment for developing web-based applications. It supports popular IDEs and tools (e.g. Eclipse), and terminal-based utilities like Vim, Git, and Docker. Most tools used on this platform are open source. A single, cross-functional development team can work effectively in this environment, especially if they are familiar with command-line workflows and containerized development. | Windows offers a wide range of development tools suitable for full-stack web application development, including Visual Studio, Visual Studio Code, IntelliJ IDEA, and Eclipse. It supports all major programming languages needed for this project. The versatility of the Windows environment allows a single development team to manage both client and server development, provided they are equipped to test across multiple platforms. | The only languages necessary at this stage are HTML, CSS, JavaScript and Jave (for the backend). In the context of mobile platforms, tools like Chrome DevTools, Safari’s Web Inspector, and mobile emulators are essential for testing responsiveness, touch interactions, and performance across various screen sizes. No platform-specific languages like Swift or Kotlin are required at this stage, so one unified development team with web expertise can handle the mobile client, keeping costs down. |

## 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**: For the server-side deployment of Draw It or Lose It, Linux is the most suitable operating platform. It is stable, secure, highly scalable, and widely supported in cloud environments. Linux distributions fall into two categories: free community-supported versions like Ubuntu Server or Debian, and commercially-supported versions like Red Hat Enterprise Linux (RHEL) and SUSE Linux Enterprise Server (SLES) – which require licensing fees in exchange for official support and service-level agreements. This flexible licensing model will allow The Gaming Room to begin with a cost-effective, open-source option and scale into a commercial version as enterprise-level support becomes necessary. Linux’s strong compatibility with AWS, Azure, and Google Cloud ensures reliable deployment and global scalability. These features make Linux ideal for hosting the real-time, multiplayer architecture needed by Draw It or Lose It.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel architecture in which core system functions – such as process scheduling, memory management, and device drivers – operate in the same memory space for speed and efficiency. Despite this structure, Linux is also modular, allowing developers to load or unload components as needed, which enhances maintainability and flexibility. This architecture is well-suited for hosting lightweight applications like *Draw It or Lose It*, which require reliable performance without the overhead of intensive graphics processing. Although the game does not rely on 3D rendering or complex simulations, it does require responsive handling of high-resolution images and timed visual effects, such as progressive de-blurring during gameplay. Linux’s asynchronous I/O capabilities and robust support for multitasking allow the system to serve multiple players and image transitions in real time with minimal latency. These characteristics ensure that the application remains smooth and responsive, even as usage scales.
3. **Storage Management**: Draw It or Lose It should rely on a combination of distributed file storage and relational database storage to support its high-resolution images, gameplay assets, and dynamic user data. For static content such as game images and UI elements, a distributed file system like Amazon Elastic File System or GlusterFS is recommended. These systems integrate seamlessly with Linux, offering scalable, fault-tolerant storage that allows multiple application servers to access shared files concurrently – a key requirement for maintaining consistent gameplay across users. To enhance performance during gameplay, some amount of image assets can be cached locally onthe user’s device, reducing latency and minimizing network overhead. In parallel, a relational database such as MySQL should be used to manage structured data, including user accounts, active game sessions, team compositions, and scoring data. This database layer ensures that the system can handle real-time updates and maintain consistent game states across users and devices. Together, this hybrid approach balances performance, scalability, and reliability – ensuring smooth gameplay while maintaining the integrity and availability of game data.
4. **Memory Management**: The Linux operating system provides robust memory management techniques that help maintain smooth performance during gameplay. It uses virtual memory, allowing the system to extend physical RAM with disk space as needed, and supports paging to load only necessary data into memory. This ensures efficient use of system resources, particularly when hosting multiple concurrent game sessions across different users The Draw It or Lose It memory footprint is modest, but the server must still track active users, game states, and temporary image assets in real time. Linux’s memory caching mechanisms help speed up access to frequently used data, while inactive resources can be automatically released to keep memory usage optimized. Additionally, Linux supports control groups – a kernel feature that allows administrators to limit memory usage for specific processes or services. This prevents a single part of the application from consuming excessive resources and ensures fair memory allocation across components, improving overall system stability.
5. **Distributed Systems and Networks**: To support cross-platform play and real-time interaction, Draw It or Lose It will operate as a distributed application with a centralized back-end server and multiple client endpoints. These clients – whether on desktop browsers or mobile devices – will connect to the server over the internet using standard web protocols. The application will rely on a combination of RESTful APIs for data transactions (such as user logins and score updates) and WebSocket connections for real-time interactions during gameplay. This hybrid communication model allows for low-latency updates and consistent synchronization across all users.

In this architecture, it is important to support session replication, where a user’s session data is shared across server instances. This allows a player to continue uninterrupted if traffic is rerouted or a server becomes unavailable. Additionally, the back-end services will be designed following stateless service principles, particularly for RESTful operations, meaning each request contains all the information needed to process it. This simplifies scaling and allows any server instance to handle incoming API calls without maintaining internal session state. To further improve reliability, the application will also include graceful reconnection handling so that temporary network drops do not force users to restart a game session. Instead, the client can attempt to reconnect and resume activity seamlessly.

1. **Security**: Security is a top priority for Draw It or Lose It, particularly given its multi-platform, internet-facing architecture. To protect user data, all communication between clients and the server will use HTTPS with TLS encryption, ensuring that sensitive information such as login credentials and gameplay activity cannot be intercepted or tampered with in transit. On the server side, access controls and role-based authentication will be enforced to prevent unauthorized access to administrative functions and user data. The application will use a secure authentication framework such as OAuth 2.0, which allows for token-based session management and minimizes the risk of session hijacking. For persistent data, encryption at rest will be applied to both databases and file systems, using the tools provided by the hosting environment (e.g., AWS Key Management Service for cloud deployments). These combined strategies help ensure that user information remains secure both in transit and at rest, regardless of which device or platform the game is accessed from.

The Linux operating system itself also provides strong security controls, including firewalls (iptables/nftables), process isolation, and mandatory access control systems like AppArmor. Regular updates, patch management, and automated vulnerability scanning will be essential to maintaining a hardened environment. These built-in features make Linux an ideal foundation for hosting secure, scalable web applications like Draw It or Lose It.