

<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 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/20/2025 | Seunghwan Lee | <Initial draft of software design document> |
| 1.1 | 10/04/2025 | Seunghwan Lee | < Replace the “Evaluation” Section (Main Task for Project Two> |
| 1.2 | 10/17/2025 | Seunghwan Lee | < Replace the “Recommendations section” Section (Main Task for Project Three> |

**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 requested to expand the Android-only game Draw It or Lose It to a multi-platform, web-based experience. The major issues that emerged were: how to ensure the ability to scale to different operating systems, retaining a unique identifier for games, teams, and players. Our architectural design is based on object-oriented principles and existing design patterns, such as a singleton to manage the game and an iterator to enforce unique names. This architecture is designed to be strong and flexible enough to allow for multiple teams, players, and deployment across platforms.

## Requirements

*The client requires the application to: (1) Support one or more teams per game, with multiple players per team; (2) Guarantee unique names for games and teams, with validation available when names are selected; (3) Allow only one instance of a game in memory at a time; (4) Assign unique identifiers to each entity including games, teams, and players; (5) Prepare the foundation for deployment on multiple platforms, including Linux, Windows, macOS, and mobile devices.*

## [Design Constraints](#_2et92p0)

The primary design constraints for the web-based distributed environment include:

- Only one instance of the GameService class should exist in memory, necessitating the Singleton pattern.

- Unique names must be enforced for games and teams across all instances, requiring careful iteration and validation.

- The system must remain lightweight and performant for multiple concurrent users, limiting the complexity of in-memory structures.

- ID generation relies on incremental counters, which must be adapted to persistent or distributed systems for production.

- Cross-platform compatibility demands use of portable languages (Java) and frameworks, avoiding OS-specific dependencies.

## [System Architecture View](#_ilbxbyevv6b6)

The Draw It or Lose It application is built using a standard multi-tier architecture. Client-side access includes web browser and mobile application connectivity to a web server which is hosting the application and exposing API endpoints. The service layer is responsible for handling all of the game logic and enforcing business rules, while the database stores specific data about the game, teams, and players. Separation of the presentation, service, and data layers allows for greater scalability of the application, improves the ability to deploy the product across different platforms, and makes future maintenance more convenient.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram demonstrates an object-oriented structure where the Entity base class defines common fields (id, name). Game, Team, and Player classes extend Entity, reusing these attributes. Game aggregates multiple Teams, and Team aggregates multiple Players, illustrating composition. The GameService class enforces uniqueness and manages lifecycle operations, applying the Singleton pattern to ensure only one instance exists. Iterator-based loops enforce name uniqueness. Key OOP principles—encapsulation, inheritance, and polymorphism—are applied to reduce redundancy and improve maintainability.

**"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)

1. Server Side: The client has asked you to create a web-based application. This implies a server-style configuration for hosting the website and allowing it to scale up to thousands of players. What does this mean for your ability to host the software application on each operating platform listed above?

Evaluate various platforms for their characteristics, advantages, and weaknesses for hosting a web-based software application. Consider the following in your evaluation and articulate your findings in the software design template:

- Does each of the operating platforms offer a server-based deployment method where the website will be hosted?

- What are the potential licensing costs to the client, The Gaming Room, for the server operating system?

2. Client Side: The client wishes to move beyond their current Android-only application to supporting players on iOS and Android mobile platforms, as well as traditional desktop-based operating systems. The application must be delivered as a modern, responsive HTML interface running inside the web browser for desktop clients (Linux, Mac, Windows), as well as on mobile platforms. Each will be capable of communicating with the back-end web application running on the server.

Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients. Consider the following in your evaluation and articulate your findings in the software design template:

- What is required of the application development process to ensure the application is compatible with all web browser platforms and mobile devices?

3. Development Tools

Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on each operating platform. Consider the following and articulate your findings in the software design template:

- What impact do these technical requirements have on a development team? Consider whether multiple development teams may be needed.

- Are there licensing costs related to the development tools?

“Evaluation” Section (Main Task for Project Two)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS can host server applications using Apache, Nginx, or Node.js, and is stable and secure. However, macOS is rarely used for large-scale production due to higher hardware costs, limited enterprise hosting support, and lower flexibility for horizontal scaling. It is suitable for staging or development environments but suboptimal for thousands of concurrent users. Licensing costs are primarily hardware-related, with minimal OS licensing fees. | Linux is the industry standard for web servers. It supports Apache, Nginx, Node.js, Docker, and Kubernetes natively, integrates seamlessly with cloud platforms (AWS, Azure, GCP), and can scale horizontally to thousands of concurrent players. Most distributions are open-source and free, resulting in low total cost of ownership. Linux provides strong security, robust performance under load, and mature enterprise tooling, making it the recommended platform for production deployment. | Windows Server supports .NET integration, GUI-based administration, and Active Directory. It is suitable for enterprise environments and integrates well with Microsoft technologies. Licensing costs (Windows Server CALs and optional Microsoft software) can increase the total cost of ownership. Performance under high concurrency is generally lower than that of Linux. It is a reasonable option if the application depends on Microsoft frameworks or infrastructure. | Mobile devices are not suitable for server hosting due to limited CPU, memory, storage, and battery capacity. Instead, iOS and Android devices function as clients that connect to the central server. They do not incur server licensing costs but require optimization for network communication and responsiveness. |
| **Client Side** | macOS desktop clients primarily use Safari and Chrome. The platform’s consistent hardware and OS reduce fragmentation, but testing is required for rendering differences, input handling, and performance. Access to Apple hardware increases testing costs, but the user experience is generally uniform and high quality. | Linux desktops access the game via Chrome, Firefox, and other browsers. Testing should cover multiple distributions (Ubuntu, Fedora, Debian) to ensure consistent rendering and functionality. Open-source development tools make deployment cost-effective, though QA effort is slightly higher due to environmental variability. | Windows dominates the desktop market. Supporting it requires testing across Edge, Chrome, and Firefox browsers. Performance and UI consistency are generally predictable. Ensuring compatibility here maximizes reach, and the broad availability of Windows machines simplifies user support. | Mobile clients require responsive web design and/or hybrid/native app development. Supporting both iOS and Android increases development complexity and testing requirements (screen sizes, touch input, OS versions). Cross-platform frameworks like React Native or Flutter help reduce duplicate effort and cost, while Progressive Web Apps (PWA) can enhance mobile experience without full native apps. |
| **Development Tools** | macOS supports IntelliJ IDEA, Eclipse, Xcode, Android Studio, and cross-platform build tools like Gradle and Maven. It enables both backend (Java, Node.js) and mobile (iOS, Android) development. Licensing costs are minimal beyond hardware; publishing iOS apps requires Apple Developer Program membership ($99/year). | Linux provides a robust environment for backend and web development, supporting Eclipse, IntelliJ IDEA (Community Edition), VS Code, Node.js, Docker, and CI/CD pipelines. Most tools are open source or free. Linux is ideal for scalable development and automated deployment. | Windows supports Visual Studio (Community/Professional), IntelliJ IDEA, Eclipse, and .NET development. IDE licensing may incur costs depending on edition. It is versatile for web, Java, and .NET projects, and user-friendly for teams familiar with Microsoft technologies. | Mobile development uses Android Studio (free) and Xcode (required for iOS builds). Cross-platform frameworks (React Native, Flutter) enable code reuse and reduce duplicated work. Device testing requires multiple physical devices or cloud-based device farms, which may add operational cost. Publishing to app stores (App Store, Google Play) introduces additional fees. |

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, especially a cloud-based distribution like Red Hat Enterprise Linux (RHEL) or Ubuntu Server, is the best operating system for hosting Draw It or Lose It. Linux is the industry standard for deploying web applications because it provides unparalleled performance, scalability, and stability in high-concurrency settings. It easily integrates with contemporary cloud providers like AWS, Azure, and Google Cloud and supports a large variety of server software, including Apache, Nginx, Node.js, and Docker.

The Gaming Room finds Linux to be the most economical long-term choice, as it is open-source and does not require the expensive licensing fees associated with proprietary operating systems. Additionally, Linux offers automated scaling and sophisticated system monitoring tools, both of which are essential for managing numerous players at once in a distributed multiplayer setting.

1. **Operating Systems Architectures**

Linux can effectively support multitasking, multiprocessing, and multiuser operations thanks to its monolithic kernel architecture enhanced by modular components. In order to minimize latency between hardware and applications, the kernel directly controls memory, file systems, device drivers, and process scheduling.

Draw It or Lose It benefits from this structure because it allows various processes, including databases, web servers, and game logic services, to operate independently at the same time. Additionally, administrators can dynamically load or unload kernel modules thanks to the modular design, which optimizes performance for particular workloads. Additionally, Linux's support for containerization (using Docker and Kubernetes) and virtualization (using KVM or Xen) allows for isolated environments for game instances, guaranteeing scalability and stability across distributed systems.

1. **Storage Management**

A hybrid storage approach is recommended to ensure data reliability, scalability, and speed. Relational databases such as PostgreSQL or MySQL should be used to manage structured data, including player profiles, game sessions, and scoring information. These databases enforce data integrity through ACID compliance, ensuring that every transaction—such as player joins or team creation—is executed consistently.

For unstructured assets, such as player drawings and images, a cloud-based object storage solution (e.g., AWS S3 or Google Cloud Storage) should be implemented. Object storage provides redundancy and global accessibility, allowing assets to be retrieved quickly regardless of platform or location. To further enhance performance, caching mechanisms such as Redis or Memcached can store frequently accessed data in memory, minimizing database load during peak activity periods.

1. **Memory Management**

Draw It or Lose It and other real-time, high-traffic applications benefit greatly from Linux's advanced memory management strategies. By using virtual memory, it shields game instances from interference or memory leaks by providing each process with its own address space. Demand swapping and paging optimize system performance under high concurrency by ensuring that only essential data is loaded into physical RAM.

In order to speed up disk I/O operations, the Linux kernel also keeps page caches. This helps with frequent data access, like leaderboard updates or user session retrieval. These memory management techniques guarantee effective resource usage, quick response times, and consistent performance for numerous concurrent users when paired with automatic garbage collection from high-level languages like Java or.NET.

1. **Distributed Systems and Networks**

A distributed client–server architecture should be used by Draw It or Lose It in order to facilitate cross-platform functionality and real-time interaction. A central Linux-based server that connects to client apps on web browsers, iOS, and Android smartphones will house the game's essential services. WebSocket connections should be used for real-time updates and live gameplay synchronization, while RESTful APIs should be used for standard data exchange.

To maintain scalability and reliability:

- Load balancers, like Nginx or HAProxy, will divide traffic equally among several servers.

- During outages, replication and failover procedures will guarantee data consistency and avoid service disruptions.

- Static content will be dispersed globally by content delivery networks (CDNs), which will lower latency for users in various geographical locations.

- Service discovery and orchestration tools (e.g., Kubernetes) will manage containerized services dynamically to ensure optimal uptime.

The system can manage thousands of connections at once thanks to its distributed structure, which also makes it resistant to hardware or network outages.

1. **Security**

For The Gaming Room to preserve user data and uphold confidence, security is crucial. HTTPS with TLS encryption should be used for all client-server communications to prevent the interception of sensitive data, including game data or login credentials.

Secure token-based systems like OAuth 2.0 or JWT (JSON Web Tokens), which allow stateless session management across distributed environments, should be used for authentication and authorization. Role-based access control (RBAC) on the server side can limit administrative actions to authorized personnel only.

Strong algorithms like AES-256 should be used to encrypt data in databases and cloud environments while it is at rest. Unauthorized access can be prevented by firewalls, intrusion detection systems (IDS), and Linux security tools like SELinux and iptables. Frequent penetration tests, code audits, and system updates will further strengthen the infrastructure against changing threats.

All of these safeguards work together to keep Draw It or Lose It safe, legal, and reliable on all platforms.

**References**

Red Hat. (2025). Linux system architecture and kernel design. https://www.redhat.com/en/topics/linux

Amazon Web Services. (2025). Introduction to cloud object storage (Amazon S3). https://aws.amazon.com/s3/features/

Microsoft. (2025). WebSocket and REST API best practices. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

Cisco Systems. (2020). Campus LAN and Wireless LAN Solution Design Guide. https://www.cisco.com/c/en/us/td/docs/solutions/CVD/Campus/cisco-campus-lan-wlan-design-guide.html