Gaming Room

*Draw It or Lose It*

# Software Design Template

Version 3.0

## Table of Contents

[Software Design Template 1](#_Toc206347974)

[Table of Contents 2](#_Toc206347975)

[Document Revision History 2](#_Toc206347976)

[Executive Summary 3](#_Toc206347977)

[Requirements 3](#_Toc206347978)

[Design Constraints 3](#_Toc206347979)

[Rationale 4](#_Toc206347980)

[Evaluation 5](#_Toc206347981)

[Recommendations 6](#_Toc206347982)

## Document Revision History

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/17/2025 | Dan Randolph | Initial Draft – Summarized problem statement and identified high level requirements, design constraints and rationale. |
| 2.0 | 07/31/2025 | Dan Randolph | Evaluate characteristics, advantages, and weaknesses of Linux, Mac, Windows and mobile platforms. |
| 3.0 | 08/14/2025 | Dan Randolph | Final recommendations with comparison of Linux vs. cloud-based hosting, updated architecture, storage, memory, distributed systems, and security analysis. |

## Executive Summary

Video game company *The Gaming Room* plans to transition their Android-based game *Draw It or Lose It* into a web-based, multi-platform application. Expanding beyond Android will improve accessibility for any users with an internet connection, drawing in attention. The team-based game is a competitive guessing game where a stock image is rendered to the screen to provide clues over multiple rounds.

Scalability and reliability will be a focal point, relying on a flexible structure that ensures stability, supports many users at once and can grow with future needs. We intend to build a reliable foundation that ensures each team, players and game sessions remain unique and secure. These up-front design decisions will simplify game management and ensure the best user experience for both players and administrators while ensuring *The Gaming Room* additional user support, new feature additions and most importantly a broader audience.

To ensure an especially stable game, game security will be a primary goal while ensuring an engaging player experience. The system will be designed to support thousands of concurrent users and be easily expandable for future updates/features.

This document also evaluates the key characteristics of Linux, Windows, Mac, and mobile platforms to support the client’s decision on how and where to deploy the application for maximum reach and efficiency. It will emphasize the evaluation of a Linux and cloud-based deployment platform couple with a final recommendation tailored to The Gaming Room’s needs.

## Requirements

Business:

* Application must be web-based with multi-user, multiplayer functionality and support
* All game, team and player names must be unique and validated upon creation
* Application must support multiple rounds of timed gameplay
* The application should be compatible across desktop and mobile browsers

Technical:

* A single instance of GameService can exist in memory at a time
* Teams consist of multiple players, while games consist of multiple teams
* Application must support entity management with proper ID tracking
* Design patterns such as Singleton and Iterator must be used

## Design Constraints

1. Web-based distributed environment: Application must support multiple clients (users) accessing the game service simultaneously, which will require consistent, centralized game state management.
2. Single instance rule: A singleton pattern will be utilized to enforce only a single instance of GameService at a time. All creation and access to games, teams and players must go through this service.
3. Unique entity naming: Naming for games, teams and players cannot be duplicated, requiring an iterator pattern to verify uniqueness at creation time.
4. Scalability and Modularity: To ensure future game improvements, code will be modularized utilizing an OOP design, allowing for future leaderboard and chat feature upgrades. Each class will maintain a single responsibility and support easy extensibility.
5. Security via Basic Authentication: For prototyping, the application uses BasicAuth to restrict access based on user roles. Future iterations can extend this with more robust methods (e.g. OAuth2 or token-based auth).

## Rationale

1. Singleton Pattern: This ensures a single point of access for managing game state across multiple sessions, while reducing complexity in managing concurrent gameplay logic.
2. Iterator Pattern: This will provide a clean and efficient method which enforces unique naming for all entity lists, so that redundant or conflicting data is avoided during game setup.
3. Entity inheritance model: Used to promote consistency in how IDs and names are handled across all objects and allows for easy debugging and scalability with minimal code duplication.
4. Designed for Distribution: Session management is supported via a centralized control paired with modular design. This prepares the system for cloud deployment with a scalable backend infrastructure.
5. Cross-Platform Architecture: Using RESTful APIs and responsive web design ensures the system is compatible with Linux, Windows, Mac, and mobile platforms without needing separate native applications.

## Evaluation

| **Development Requirements** | **Linux** | **Windows** | **Mac** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Linux is widely adopted for web server hosting and supports Apache, Nginx, and Tomcat. It offers superior scalability, is open source, and has no licensing costs, making it ideal for server-side deployments. | Windows supports IIS and Apache but generally has higher resource usage. It requires a license, increasing costs for The Gaming Room. IIS offers strong .NET integration but less flexibility than Linux. | Mac can run Apache or Nginx but is rarely used in production environments. It's Unix-based and reliable but lacks the widespread deployment support found in Linux. Hosting costs may be higher due to Apple hardware and licensing. | Mobile devices are not suitable for hosting web applications. They are client endpoints only, so server-side deployment must happen on another platform. |
| **Client Side** | Linux users typically access web apps through Chrome or Firefox. Testing is simple and development effort is low if apps are built with standard HTML5 and JavaScript. | Windows users use Chrome, Firefox, and Edge. Compatibility is high, though Edge may require specific style testing. Development time remains low due to widespread standard browser support. | Mac fully supports all modern browsers. Development requires testing in Safari, which may handle some styles and scripts differently. Building for Mac desktop browsers is straightforward if using responsive design. | Mobile browsers differ significantly across Android and iOS. Responsive design using HTML5/CSS3/JS is required. iOS Safari has tighter security and layout differences; Android/Chrome is more permissive. Development time and testing increases on mobile. |
| **Development Tools** | Most Linux development tools are free and open source. Common tools include Eclipse, IntelliJ, and VS Code, with languages like Java, Python, and Node.js readily supported. | Windows supports Visual Studio (which may require a license) and open-source tools like Eclipse and VS Code. Development in Java, .NET, and JavaScript is well supported. | Tools include Xcode for native Mac/iOS development and IntelliJ or VS Code for web development. Most are free, but Xcode is Mac-only and tied to Apple’s ecosystem. | Mobile development requires Android Studio and Xcode (for iOS). Web wrappers may also be created using VS Code with frameworks like React or Ionic. Cross-platform support increases complexity. |

## Recommendations

Operating Platform  
After evaluating both Linux-based on-premises hosting and cloud-based solutions (AWS, Azure, Google Cloud), the recommended platform is cloud-based deployment using a Linux server environment. This hybrid choice combines the scalability, redundancy, and managed services of cloud hosting with the cost-effectiveness, flexibility, and reliability of Linux. While a dedicated Linux server could meet requirements, cloud infrastructure offers near-instant scaling, global availability, and reduced maintenance burden.

Operating System Architectures  
The chosen cloud platform will run Linux on x86\_64 or ARM architectures. This modular Unix-like OS is optimized for multi-user, multi-tasking environments, making it well-suited for hosting web applications. Virtualization and containerization (e.g., Docker, Kubernetes) will allow for efficient resource allocation and simplified deployment pipelines.

Storage Management  
The 200 high-definition image files (~1.6GB total) will be stored in cloud object storage (e.g., AWS S3 or Azure Blob Storage) for durability and low-latency retrieval. Metadata, player data, and game state will reside in a managed relational database service (e.g., Amazon RDS or Azure Database for PostgreSQL). This ensures data integrity, backups, and global accessibility with minimal administrative overhead.

Memory Management  
Cloud-hosted Linux instances will use demand paging, virtual memory, and caching strategies to ensure rapid rendering of images. Containers can be allocated sufficient RAM for concurrent sessions, and content delivery networks (CDNs) will cache images near users to improve load speed. Scaling memory resources up or down will be handled automatically based on real-time traffic.

Distributed Systems and Networks  
The application will follow a microservices architecture deployed across multiple cloud regions. A RESTful API over HTTPS will facilitate communication between the game client and backend services. Load balancers will distribute traffic, while failover instances maintain uptime during outages. Network latency will be minimized using edge servers and CDNs.

Security  
All communication will be encrypted via TLS. User authentication will start with BasicAuth for prototyping but will move to OAuth2 with token-based sessions for production. Cloud security tools, including Web Application Firewalls (WAF) and DDoS protection, will guard against common attacks. Linux’s built-in security modules (SELinux/AppArmor) will provide process-level isolation.

Final Recommendation: Deploy *Draw It or Lose It* in a cloud-based environment running Linux, leveraging managed storage, scalable compute, and global network infrastructure. This approach ensures high availability, supports rapid scaling, and reduces operational complexity while keeping costs predictable.