

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 4**

**Evaluation 5**

**Recommendations 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/21/25 | Prince Larkins | <Brief description of changes in this revision> |
| 1.1 | 2/3/25 | Prince Larkins | Updated to provide a complete template |
| 1.2 | 2/18/25 | Prince Larkins | Revised template to meet requirements |
| 1.3 | 2/19/25 | Prince Larkins | Revised to meet Project 3 requirements |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room seeks to expand Draw It or Lose It beyond Android to a multi-platform, web-based application. This requires transitioning from a mobile-only app to a scalable, distributed system capable of handling thousands of concurrent users. This document evaluates different operating platforms, addressing server hosting, client-side compatibility, development tools, security, and system performance. The recommendations provided will help ensure an optimized deployment that meets the client’s requirements.

## Requirements

* **Multi-platform compatibility:** Ensure the game is accessible via browsers (Chrome, Edge, Firefox, Safari) on Windows, macOS, Linux, and mobile devices.
* **Scalability:** Must support thousands of users via cloud infrastructure and load balancing.
* **Security:** Implement OAuth 2.0, JWT authentication, SSL encryption, and role-based access control (RBAC).
* **Efficient storage & rendering:** Optimize caching mechanisms to minimize memory and bandwidth usage.
* **Server-client communication:** Ensure seamless real-time updates using REST APIs and WebSockets.

## [Design Constraints](#_2et92p0)

* Transition from a mobile app to a web-based distributed system.
* Large-scale storage for high-resolution images (1.6GB requirement).
* Security concerns related to multi-user interactions.
* Performance considerations for rendering real-time game elements.

## [System Architecture View](#_ilbxbyevv6b6)

The client-server architecture allows the game to run on multiple devices while keeping core functionality on the backend server.

Client Side (Web & Mobile App):

* The game UI runs in a browser or mobile app.
* Communicates with the server via REST API requests.
* Implements real-time updates using WebSockets (optional).

Server Side (Backend & Database):

* Processes game logic and manages game state.
* Stores user accounts, game records, and team assignments.
* Handles authentication and security enforcement.

Database & Storage:

* Holds game history, user progress, and images.
* Uses caching mechanisms to improve performance.

## [Domain Model](#_8h2ehzxfam4o)

**Object-Oriented Programming (OOP) Principles Used:**

* **Abstraction:** Entity class defines shared attributes.
* **Encapsulation:** Private attributes in Game, Team, and Player.
* **Inheritance:** Game, Team, and Player extend from Entity.
* **Polymorphism:** Method overloading allows flexible retrieval of 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)

**Server-Side Hosting**

| **Development Requirements** | **Hosting Capabilities** | **Licensing Costs** | **Virtualization** | **Cloud Support** |
| --- | --- | --- | --- | --- |
| **Linux** | Best for scalability, security, and cost-effectiveness. Supports Apache, Nginx, Docker, Kubernetes. | Open-source (free). | Supports Docker, Kubernetes, and VM-based deployments. | Compatible with AWS, GCP, Azure. |
| **Windows** | Enterprise-grade hosting with full .NET support. | Requires paid licensing (costly). | Supports Hyper-V and VMWare. | Strong Azure support, integrates with Microsoft products. |
| **Mac** | Rarely used for server hosting, primarily for development. | Paid licensing required. | Supports virtualization via Parallels. | Limited cloud hosting support. |
| **Mobile** | Cannot be used as a server platform. | N/A | N/A | Cloud-based storage and backend hosting (Firebase, AWS). |

**Client-Side Development Considerations**

|  |  |
| --- | --- |
| Aspect | **Details** |
| **Browser Compatibility** | Must support **Chrome, Firefox, Edge, Safari** across platforms. |
| **Client Popularity** | Windows dominates desktop installs; mobile usage is higher in consumer markets. |
| **Mobile vs. Desktop** | UI must be **responsive**, using **Progressive Web Apps (PWA)** for mobile adaptation. |
| **Development Frameworks** | **React.js, Angular.js** recommended for front-end UI. |
| **Security** | **OAuth 2.0, SSL/TLS encryption, input validation** required. |

**Development Tools and Programming Languages**

|  |  |  |  |
| --- | --- | --- | --- |
| Platform | IDEs & Tools | Languages | Licensing |
| **Linux** | VS Code, Eclipse, IntelliJ (open-source). | Java, Python, JavaScript (Node.js). | Free |
| **Windows** | Visual Studio, JetBrains (paid). | .NET, C#, JavaScript. | Paid |
| **MacOS** | Xcode (free, Apple-exclusive). | Swift, Objective-C. | Free but restricted to Apple ecosystem. |

**Security & Virtualization**

* Linux & MacOS are UNIX-based, but only Linux is open-source.
* Windows is proprietary, requiring additional licensing.
* Linux supports containerization (Docker, Kubernetes), while Windows favors Hyper-V.

## Recommendations

**Operating Platform Recommendation**

The recommended operating system for hosting Draw It or Lose It is Linux due to its scalability, cost-effectiveness, and strong security features. Linux is an industry standard for cloud-based deployments and supports containerization (Docker, Kubernetes), making it ideal for handling multiple game sessions simultaneously.

**Operating System Architecture**

The Linux operating system architecture is based on a monolithic kernel, meaning all core functionalities, including process management, memory management, and system calls, are executed in a single space. Key features include:

* **Multithreading and Scheduling**: Linux uses Completely Fair Scheduler (CFS) to efficiently distribute resources among game processes.
* **File System Support**: Linux supports ext4, XFS, and Btrfs, providing reliable storage and faster data access.
* **Process Management**: Uses Preemptive multitasking to optimize CPU cycles for handling multiple players.
* **Security Model**: Implements SELinux and AppArmor for enforcing mandatory access controls and preventing unauthorized access.

### ****Storage Management****

### For efficient storage management, the following strategies are recommended:

* **Cloud Object Storage**: Use Amazon S3 or Google Cloud Storage for scalable, redundant storage of game assets and user data.
* **Block Storage**: Utilize Amazon EBS (Elastic Block Store) or SSDs for faster access to frequently used data.
* **Database Indexing**: Optimize database queries with B-tree and Hash Indexing to speed up retrieval of game records.
* **Data Redundancy & Fault Tolerance**: Implement RAID configurations and automated backups to ensure system reliability.

**Memory Management**

The Linux memory management system efficiently handles processes through:

* **Virtual Memory & Paging**: Ensures efficient allocation of memory to active processes using demand paging and swap space.
* **Garbage Collection**: For the backend, Java (Garbage-First GC) and Node.js (V8 Engine GC) optimize memory allocation and cleanup.
* **Caching Optimization**: Utilize Redis and Memcached to store frequently accessed game data and minimize database lookups.
* **LRU (Least Recently Used) Caching**: Helps manage memory consumption by removing old game assets from cache dynamically.

### ****Distributed Systems & Network Considerations****

### To support **multi-platform gameplay**, Draw It or Lose It will be deployed as a **distributedsystem** using the following architecture:

* **Microservices-Based Deployment**: Separates core functions (game logic, authentication, storage) into independent services.
* **Load Balancing**: Implements NGINX or HAProxy to evenly distribute player traffic across servers.
* **Networking Protocols**: Uses WebSockets for real-time gameplay updates and REST APIs for player authentication and game session management.
* **Fault Tolerance**: Auto-scaling infrastructure ensures high availability even during server outages or increased traffic.

**Security Implementation**

Security is critical to protecting user data and ensuring fair gameplay. The following measures will be implemented:

* **User Authentication**:
  + Implement OAuth 2.0 & JWT (JSON Web Tokens) for secure login sessions.
  + Enforce Multi-Factor Authentication (MFA) to enhance security.
* **Data Encryption**:
  + Use TLS 1.3 encryption to protect player communications.
  + Store sensitive user data using AES-256 encryption.
* Role-Based Access Control (RBAC):
  + Define user roles (Player, Admin, Moderator) with minimum privilege policies.
* DDoS Protection:
  + Deploy Web Application Firewalls (WAFs) and rate-limiting mechanisms to mitigate attacks.
* Secure Session Handling:
  + Implement secure cookie policies and automatic session expiration for idle users.

**Conclusion**

By leveraging Linux-based hosting, distributed architecture, and advanced security measures, Draw It or Lose It can expand seamlessly across multiple platforms. The proposed microservices design, optimized memory management, and cloud-based storage solutions ensure high performance, scalability, and security.

**References**

*Silberschatz, A., Galvin, P. B., & Gagne, G. (2012). Operating system concepts (8th ed.). Wiley.*

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

Microsoft Azure. (2023). *Cloud computing services*. Retrieved from https://azure.microsoft.com

*OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. (n.d.). https://owasp.org/

Google for Developers. (n.d.). *Google for Developers - from AI and Cloud to Mobile and Web*. https://developers.google.com/

Ratliff, S. (2025, January 23). *Docker: Accelerated Container Application Development*. Docker. https://www.docker.com/

Chapel, J. (2021, December 10). Cloud Storage Cost Comparison: AWS vs. Azure vs. Google. *Medium*. https://jaychapel.medium.com/cloud-storage-cost-comparison-aws-vs-azure-vs-google-844dfff3d324