

Draw It or Lose It Software Application

# **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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/16/2024 | Daniel Hernandez | Initial draft of the software design document. |
| 1.0 | 11/30/2024 | Daniel Hernandez | Revised Table |
| 1.0 | 12/15/2024 | Daniel Hernandez | Revised Recommendations |

**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 game, *Draw It or Lose It*, into a web-based distributed environment. The software design must support scalability, cross-platform compatibility, and ensure seamless user experiences across multiple devices. To meet these goals, the design leverages a distributed system architecture using object-oriented programming (OOP) principles for maintainable and extensible code. This document outlines the requirements, constraints, architecture, and recommended platforms to ensure successful implementation.

## Requirements

The client requires the following:

1. A scalable web-based application to support multiple concurrent users.
2. Cross-platform compatibility across desktops (Windows, Mac, Linux) and mobile devices.
3. Secure user authentication and data protection across platforms.
4. Efficient use of memory and storage for optimal performance.

## [Design Constraints](#_2et92p0)

Developing the application in a web-based distributed environment poses several constraints:

1. **Platform Dependency**: The design must ensure compatibility across diverse platforms, requiring careful abstraction in the codebase to minimize OS-specific dependencies.
2. **Scalability**: The architecture must support adding new players, teams, and games without significant refactoring or performance degradation.
3. **Network Latency**: Communication between distributed systems must be optimized to minimize latency, requiring robust APIs and load balancers.
4. **Security**: The design must comply with data protection standards (e.g., encryption and secure authentication mechanisms).

These constraints demand careful planning in system architecture and technology selection to ensure reliability and usability.

## [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 demonstrates an object-oriented design where inheritance and composition are key principles.

* **Entity Class**: A base class that encapsulates common attributes (id and name) shared by all game components.
* **Inheritance**: Specific entities like Game, Team, and Player inherit from Entity, adhering to the *DRY (Don't Repeat Yourself)* principle by centralizing common behaviors.
* **Composition**: The Game class contains a list of Team objects, and each Team contains a list of Player objects, representing a hierarchical relationship between game components.
* **Singleton Pattern**: The GameService class ensures that only one instance manages the game's state, preventing conflicts in a distributed environment.

This domain model supports scalability by organizing data relationships logically and modularly.

**"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** | |  |  |  | | --- | --- | --- | | |  | | --- | | Stable UNIX-based OS, well-suited for lightweight servers but higher hardware costs. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Open-source and highly scalable; ideal for web servers. |  |  | | --- | |  | | |  | | --- | | Familiar for developers but incurs higher licensing costs. |  |  | | --- | |  | | |  | | --- | | Limited server hosting capabilities; primarily client-side functionality. |  |  | | --- | |  | |
| **Client Side** | |  | | --- | | Requires specific tools (e.g., Xcode); higher costs but good for native iOS development. |  |  | | --- | |  | | |  | | --- | | Supports responsive web design; low development costs. |  |  | | --- | |  | | |  | | --- | | Requires compatibility testing for legacy systems. |  |  | | --- | |  | | |  | | --- | | Demands responsive design to work across iOS and Android. |  |  | | --- | |  | |
| **Development Tools** | |  | | --- | | Tools: Xcode, Swift. Additional licenses required. |  |  | | --- | |  | | |  | | --- | | Tools: Visual Studio Code, Eclipse; cost-effective. |  |  | | --- | |  | | |  | | --- | | Tools: Visual Studio, .NET framework; robust but costly. |  |  | | --- | |  | | Tools: Android Studio, Xcode; recommend cross-platform frameworks like Flutter. |

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 hosting the *Draw It or Lose It* application in a distributed environment, Linux is the recommended operating platform. Linux is an open-source, highly scalable, and cost-effective platform that provides robust server capabilities. It is widely used for web server applications, offering high performance, reliability, and excellent community support. Additionally, its lightweight nature ensures efficient resource utilization, making it ideal for a scalable and distributed architecture.
2. **Operating Systems Architectures**: Linux operates on a monolithic kernel architecture, which combines core operating system functions like file management, memory management, and process management into a single layer. This design ensures minimal overhead and fast communication between processes. The architecture also supports modularity through loadable kernel modules, enabling customization for the specific requirements of *Draw It or Lose It*. This adaptability allows developers to optimize system performance by tailoring the kernel to the application’s needs.
3. **Storage Management**: For storage, Network Attached Storage (NAS) with a Linux-supported file system such as EXT4 or XFS is recommended. These file systems are robust, support large file sizes, and ensure data integrity. Additionally, Linux-based NAS can support distributed file systems like GlusterFS or Ceph, which enable high availability and scalability. These systems can handle the dynamic storage requirements of a growing user base, ensuring that saved game data, user preferences, and logs are managed efficiently.
4. **Memory Management**: Linux’s memory management system uses virtualmemory and advanced techniques like paging and demand paging to allocate resources efficiently. For *Draw It or Lose It*, the kernel dynamically allocates memory to running processes while maintaining memory isolation to prevent security vulnerabilities. The system also utilizes shared memory for inter-process communication (IPC) to ensure smooth collaboration between the different components of the application. Additionally, the Linux Out-of-Memory (OOM) killer monitors and handles memory exhaustion to maintain application stability.
5. **Distributed Systems and Networks**: To enable communication across platforms, *Draw It or Lose It* can leverage RESTful APIs hosted on a Linux-based server. These APIs provide a lightweight, platform-independent communication layer for data exchange between devices. Additionally, load balancers can distribute incoming traffic among multiple servers to prevent downtime during high usage periods. A distributed file system ensures that user data remains accessible across servers. Using containerization tools like Docker facilitates deployment consistency across platforms, while Kubernetes orchestrates container management for scalability.
6. **Security**: To ensure user data protection, Linux provides several security features:

* **File Permissions and Access Controls**: Enforce strict read, write, and execute permissions for data access.
* **Encryption**: Use tools like **OpenSSL** for secure data transmission and **LUKS** for disk encryption to protect data at rest.
* **Firewall Protection**: Use **iptables** or **UFW** to secure server communication.
* **Authentication and Authorization**: Implement multi-factor authentication (MFA) and robust user authentication protocols to prevent unauthorized access.