

# 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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 12/11/24 | Gonzalo Ramos | Final Draft |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room seeks to expand its current Android-only game, Draw It or Lose It, into a web-based application that supports multiple platforms. This expansion will require a robust, scalable, and distributed architecture that enables the game to run seamlessly across platforms, ensuring unique instances for teams and players. Our proposed solution involves implementing object-oriented programming (OOP) principles, leveraging software design patterns, and utilizing an efficient storage and networking system to support distributed communication. The design ensures scalability, platform compatibility, and data security, meeting the client’s technical and business requirements.

## Requirements

The game application must meet the following requirements:

1. Support one or more teams per game.
2. Allow multiple players per team.
3. Ensure unique names for games, teams, and players.
4. Maintain only one instance of the game in memory at any time.
5. Enable communication and synchronization across multiple platforms.

## [Design Constraints](#_2et92p0)

Developing the game in a web-based distributed environment introduces the following constraints:

1. Scalability**:** The system must support multiple users and platforms without performance degradation.
2. Data Integrity**:** Unique identifiers for games, teams, and players must be maintained to avoid duplication.
3. Platform Compatibility**:** The design must accommodate different operating systems, including Windows, Mac, Linux, and mobile devices.
4. Latency: Network communication between distributed systems must be optimized to ensure real-time synchronization.
5. Security: User data and communication must be encrypted and protected from unauthorized access. These constraints affect the system’s architecture, requiring efficient algorithms, secure protocols, and robust storage solutions.

These constraints affect the system’s architecture, requiring efficient algorithms, secure protocols, and robust storage solutions.

## [System Architecture View](#_ilbxbyevv6b6)

The system architecture for Draw It or Lose It involves a client-server model where a centralized server hosts game logic and databases while clients (web and mobile devices) interact through APIs. The architecture ensures separation of concerns and efficient scalability.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram depicts the relationships between the game, team, and player entities. Each game can have multiple teams, and each team can have multiple players. The GameService class manages the creation and retrieval of these entities using the singleton design pattern to ensure only one instance of the service exists in memory. The Entity base class demonstrates inheritance, encapsulating shared attributes (id and name) for all derived classes (Game, Team, and Player). This OOP approach enhances code reusability, scalability, and 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)

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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | |  | | --- | |  |  |  | | --- | | Mac systems are stable but expensive. Not widely used for hosting web servers. | | Linux is reliable, scalable, and widely used for hosting. Open source and cost-effective. | Windows servers offer ease of use but are more expensive than Linux. | Not suitable for hosting a server-side application due to limited resources and platform-specific constraints. |
| **Client Side** | Mac systems are stable but expensive and not commonly used for hosting large-scale web applications. They provide excellent performance for smaller-scale hosting but are limited in scalability compared to Linux. | Linux is the most widely used platform for hosting web-based applications. It is open-source, highly scalable, and cost-effective, with extensive community support for server-side deployment. | Windows servers are user-friendly and offer seamless integration with Microsoft tools. However, they are more expensive due to licensing costs and less flexible than Linux. | Mobile devices are not suitable for hosting server-side applications due to hardware constraints and lack of server-grade resources. |
| **Development Tools** | Mac clients are favored by creative professionals but require specialized development expertise for compatibility. The market share for Mac clients is growing steadily. | Linux clients are less common among general users but are widely used in technical and educational settings. Development for Linux is cost-effective and requires minimal resources. | Windows dominates the client-side market, offering extensive compatibility and a wide user base. Development is straightforward and well-supported, though slightly costlier than Linux. | Mobile clients require platform-specific development for iOS and Android. Cross-platform tools like Flutter or React Native can reduce costs but still demand significant time and expertise. |

## Recommendations Operating Platform: Linux remains the top recommendation for server-side hosting due to its scalability, cost-effectiveness, and strong ecosystem for web hosting. It meets all client requirements for a robust and scalable backend.

## Operating Systems Architectures: A client-server architecture with a Linux-based backend and responsive HTML-based clients ensures broad compatibility. Use modern frameworks like React or Angular to ensure a seamless experience on all platforms.

## Storage Management: Implement a relational database like MySQL for efficient handling of game data. This allows scalability and ensures consistency across distributed systems.

## Memory Management: Dynamic memory allocation for active game instances coupled with caching for frequently accessed data will optimize server performance.

## Distributed Systems and Networks: Use REST APIs for communication between clients and servers. For real-time features, integrate WebSocket protocols for low-latency interactions.

## Security: Use HTTPS and robust encryption protocols to secure communication. OAuth 2.0 for authentication and RBAC for data protection will address security concerns effectively. Conduct regular security audits to mitigate potential vulnerabilities.

1. Operating Platform: Linux is the recommended operating platform for hosting the server due to its scalability, reliability, cost-effectiveness, and wide adoption in web hosting environments.
2. Operating Systems Architectures: Use a client-server architecture with a Linux-based backend server and clients running on Windows, Mac, and mobile devices. This ensures broad compatibility and high performance.
3. Storage Management: Implement a relational database management system (RDBMS) such as MySQL or PostgreSQL to manage game data. These databases provide robust support for scalability and distribution systems.
4. Memory Management: The server will use garbage collection for memory optimization and allocate memory dynamically to active instances of games, teams, and players. Efficient caching mechanisms can be employed for frequently accessed data.
5. Distributed Systems and Networks: Communication between platforms can be achieved using REST APIs or WebSocket protocols. These protocols enable real-time data synchronization and low-latency communication between clients and the central server. Redundant systems can be implemented to handle connectivity issues or server outages.
6. Security: Use HTTPS for secure communication and encrypt all user data stored in the database. Implement OAuth 2.0 for user authentication and role-based access control (RBAC) to restrict access to sensitive game data. Employ firewalls and regular penetration testing to ensure robust protection against cyber threats.