

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 |
| --- | --- | --- | --- |
| 1.0 | 6/2/2024 | Erika Rodriguez | Filling out requirements |

**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 is transitioning from an Android-only game application, “Draw It or Lose It”, to a web-based game that supports multiple platforms. This document outlines the software design and development plan for this transition. The main goal is to create a scalable, efficient, and user-friendly web application that maintains the core functionality of the original game while expanding its accessibility. This includes implementing unique identifiers for games, teams, and players, ensuring unique names for game and team instances, and maintaining a single instance of the game in memory using design patterns like singleton an iterator.

## Requirements

Business Requirements

• Develop a web-based version of “Draw It or Lose It”

• Support multiple platforms

• Maintain ore gameplay mechanics

Technical Requirements  
• Implement unique identifiers for games, teams, and players.

• Ensure game and team names are unique

• Maintain a single instance of game in memory

## [Design Constraints](#_2et92p0)

Developing a web-based distributed game application imposes several design constraints:  
• Concurrency: Handling multiple users accessing and interacting with the game simultaneously requires robust concurrency model.

• Scalability: The system must be scalable to accommodate increasing numbers of users and games without performance degradation

• Data Consistency: Ensuring the game state and data remain consistent across different sessions and users  
• Latency: Minimizing latency to provide a smooth and responsive user experience  
• Security: Protecting user data and preventing unauthorized access

## [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)

• Entity Class: A base class with attributes ‘id’ and ‘name’. Common methods include constructors ‘getId()’, ‘getName()’, and ‘toString()’.

• Game Class: Inheritys from ‘Entity’. Contains a list of ‘Team’ objects and methods to add a team and convert game information to a string.

• Team Class: Inheritys from ‘Entity’. Contains a list of ‘Player’ objects and methods to add a player and convert team information to a string.

•Player Class: Inherits from ‘Entity’. Includes a constructor and a ‘toString()’ method.

•GameService Class: Manages the list of games, next identifiers for games, teams, and players, and provides methods to add and retrieve games. It uses the Singleton pattern to ensure only one instance exists.

**"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** | Mac servers are known for their stability and strong performance. However, they are costlier and less common in the industry compared to Linux servers, leading to potential challenges in finding support and resources. | Linux servers are preferred for hosting web applications due to their robustness, security, and cost effectiveness. They offer high performance and extensive support for various web technologies. | Windows Servers provide good performance and are compatible with many enterprise applications. However, they can be more expensive than Linux servers and may require more maintenance. | Hosting directly on mobile devices is impractical due to limited resources and security concerns. Instead, mobile clients should connect to a centralized web server. |
| **Client Side** | Developing for Mac clients requires expertise in MacOS-specific development and testing, potentially increasing costs and development time. | Supporting Linux clients is generally less resource-intensive and cost-effective. Linux offers excellent development tools and flexibility. | Developing for Windows clients requires more consideration of various versions and ensuring compatibility. It can be more costly but offers a large user base. | Supporting mobile devices involves developing responsive designs and ensuring compatibility with different operating systems (iOS, Android). It increases development time and costs but is crucial for broad accessibility. |
| **Development Tools** | Xcode, IntelliJ IDEA, and Visual Studio Code are popular tools. Programming languages include Swift, Java, and JavaScript. | Eclipse, IntelliJ IDEA, and Visual Studio Code are common IDEs. Languages include Java, Python, and JavaScript. | Visual Studio Code, IntelliJ IDEA, and Visual Studio Code are widely used. Languages include C#, Java, and JavaScript. | Android Studio for Android, Xcode for iOS, and cross-platform tools like Flutter and React Native. Languages include Java, Swift, Dart, and JavaScript. |

## 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 Is recommended as the primary operating platform for its robustness, security, cost-effectiveness, and extensive support for web technologies. Linux offers high performance and stability. Making it ideal for hosting a web-based game application like “Draw It or Lose It.”
2. **Operating Systems Architectures**: Linux supports multiple architectures, including x86 and ARM, which are widely used in servers and embedded in systems. It provides a modular and scalable architecture, allowing for efficient resource management and adaptability to various hardware configurations. The Linux kernel’s design enables it to handle multiple processes simultaneously, making it suitable for a distributes environment where concurrency and parallelism are crucial.
3. **Storage Management**: Implement a relational database management system (RDBMS) like PostgreSQL for storage management. PostgreSQL is known for its reliability, data integrity, and robustness. It supports complex queries and transactions, ensuring that all game-related data, such as user accounts, game states, and scores, are consistently and securely managed. Additionally, PostgreSQL offers advanced features such as indexing, replication, and backups, which are essential for maintaining data integrity and availability.
4. **Memory Management**: Linux employs efficient memory management techniques, such as paging, swapping, and caching, to optimize the use of available memory. Paging divides memory into fixed-sized pages, allowing the operating system to manage memory more effectively. Swapping enables the system to move inactive pages to disk, freeing up RAM for active processes. Caching improves performance by storing frequently accessed data in fast-access memory. These techniques ensure that “Draw It or Lose It” can handle large numbers of concurrent users and maintain smooth performance even under heavy load.
5. **Distributed Systems and Networks**: To enable communication between different platforms, use RESTful APIs. REST (Representational State Transfer) is a lightweight and scalable architecture that allows different components of the system to interact over HTTP. By designing RESTful APIs, the game application can facilitate communication between the server and clients on various platforms (web, mobile, etc.). Ensure reliable connectivity by implementing load balancing and failover mechanisms to handle outages and maintain service availability. Use content delivery networks (CDNs) to reduce latency and improve the user experience.
6. **Security**: Security is paramount for protecting user information and maintaining trust, Implement HTTPS to encrypt data transmitted between the client and server, preventing eavesdropping and man-in-the-middle attacks. Use strong encryption protocols and secure authentication mechanisms, such as OAuth or JWT (JSON Web Tokens), to protect user accounts and sessions. Regularly update the system and apply security patches to address vulnerabilities. Implement firewalls and intrusion detection/prevention systems (IDS?IPS) to safeguard against unauthorizes access and cyberthreats. Ensure compliance with data protection regulations (e.g. GDPR, CCPA) to protect user privacy and data integrity.