

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_gjdgxs)

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 2.0 | <mm/dd/yy> | Zayn Khan | In the evaluation, I focused on programming languages on each platform, the pros and cons from the server side and client side, and the development cost |
| 3.0 |  | Zayn Khan | In this version, I have updated the recommendations and went into detail to explain each recommendation to ensure we are able to develop a quality product |

**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](#_35nkun2)

"Draw It or Lose It" is a web-based game where multiple teams compete to guess a puzzle based on rendered drawings. The game will have four rounds with unique puzzles, and only one instance of the game will be maintained at any given time. This project aims to develop a scalable, distributed web-based solution that adheres to software design principles such as the Singleton pattern, ensuring that game and team names are unique and using object-oriented principles for code maintainability.

## Requirements

* *Only one instance of a game should exist at a time.*
* *The game supports multiple teams, and each team can have multiple players.*
* *Each game must have a unique name.*
* *Team names must be unique within each game but can be reused across different games.*
* *only one instance of the game service is running in memory.*

## [Design Constraints](#_1ksv4uv)

1. Since the application is hosted online the application should be designed for distributed environments. this means the management of resources and communication between server and client must be taken into account for quality and performance.
2. the game must support real-time updates and handle multiple current users
3. only one instance of the game can exist in memory at any given time to manage the game instances
4. unique names
5. Cross platform capability the application will be used from different platforms.
6. security and data integrity needs secure communication between client and server
7. scalability must be able to support the number of users and games

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

The diagram presents the structure of a game application, employing object-oriented principles such as inheritance, composition, and the singleton pattern. The Entity class acts as a base class, sharing common attributes like id and name with the Game, Team, and Player classes, thereby promoting code reuse and eliminating redundancy. The GameService class, designed following the singleton pattern, ensures that only one instance manages the Game's logic, from creating and retrieving Game, Team, and Player instances. This pattern ensures centralized control, preventing conflicts within the Game's lifecycle. The diagram also demonstrates the composition, with each Game containing multiple Team objects, and each Team containing multiple Player objects. This hierarchical relationship ensures that teams and players are tightly coupled to their respective games, meeting the requirements for team-based gameplay. The ProgramDriver class, serving as the application's entry point, guides the flow of the application, while SingletonTester verifies that the GameService class adheres to the singleton design pattern, ensuring efficient game state and scalability management.

**"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](#_z337ya)

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** | Pros: Good for Java, Python, and C++ web apps, integrates well with iOS.  Cons: High hardware cost. | Pros: Best for scalable web applications, supports Java, Python, and C++. Low cost and highly customizable. Cons: Requires expertise to manage. | Pros: Compatible with Java, Python, and C++, strong enterprise support.  Cons: High licensing cost, less lightweight than Linux for large-scale apps. | Pros: Ideal for client-server models with server-side logic in Java, Python, or C++. Cons: Limited server hosting; more focus on client-side apps. |
| **Client Side** | Pros: Good for developing cross-platform web and macOS/iOS apps with Java/Python.  Cons: Additional work for iOS native apps. | Pros: Easy web development with Java/Python across platforms and open-source tools.  Cons: Complex setup for native desktop apps. | Pros: Good tool support for Java/Python/C++, strong Windows compatibility. Cons: Licensing fees, and less streamlined for web apps compared to Linux. | Pros: Java (Android) and Python are well-supported for server integration. Cons: Native mobile development requires platform-specific expertise (Java for Android, Swift for iOS). |
| **Development Tools** | Tools: IntelliJ, PyCharm, Xcode. Licensing: Free IDEs, but costly hardware. | Tools: IntelliJ, Eclipse, PyCharm. Licensing: Mostly free and open-source tools. | Tools: IntelliJ, Visual Studio, PyCharm. Licensing: Paid Windows licenses, but open-source options are available. | Tools: Android Studio (Java), Xcode (iOS). Licensing: Both IDEs are free, but iOS requires a paid account for distribution. |

## Recommendations

Analyze the characteristics and techniques specific to various systems architectures and make a recommendation for The Gaming Room. Specifically, address the following:

1. **Operating Platform**: Linux is the recommended choice for the server platform. With its scalability, flexibility, and strong support for web applications, Linux is the ideal platform as "Draw It or Lose It" expands to multiple platforms. Its open-source nature makes it a cost-effective solution for hosting a web-based game. Linux's widespread use for server hosting is a testament to its stability and security. It supports critical development languages such as Java, Python, and C++, which are integral to the game's architecture. Furthermore, it seamlessly integrates with cloud platforms, ensuring high scalability for growing user bases.
2. **Operating Systems Architectures**: Linux operates on a design where core services like memory management, file system management, and process scheduling are all bundled together in a single large kernel. This setup allows for efficient communication between processes and hardware. Combined with its open-source nature, it provides better control over customization and performance optimization. As a bonus, Linux supports various file systems, making it adaptable to different storage needs.
3. **Storage Management**: For the storage management of "Draw It or Lose It," Cloud Storage is recommended. Cloud storage offers scalability, allowing the game's asset library to expand seamlessly as needed while providing global accessibility, ensuring fast asset retrieval regardless of the server location. It is a cost-effective solution with pay-as-you-go pricing, reducing the need for significant upfront investments in physical hardware. Additionally, cloud storage ensures data reliability and redundancy, safeguarding the game's assets and providing disaster recovery in case of server failures. This approach offloads the storage burden from local servers, optimizing performance and ensuring smooth gameplay.
4. **Memory Management**: Linux employs advanced memory management techniques such as virtual memory and demand paging, ensuring efficient resource allocation. Virtual memory allows processes to use more memory than is physically available by temporarily swapping unused data to disk. For "Draw It or Lose It," Linux can allocate memory dynamically, meaning the game will only use the memory it needs at any given time. By employing memory caching and buffering, the system can handle multiple users and requests without sacrificing performance.
5. **Distributed Systems and Networks**: A distributed system architecture is necessary for "Draw It or Lose It" to operate across various platforms. This includes deploying web-based services that can interact with mobile and desktop clients via APIs. A message queue system that can be integrated for low-latency data exchanges between servers and clients to manage real-time gameplay and interactions.
6. **Security**: The most critical aspect is protecting user data. Linux offers many security options, such as SSL/TLS encryption and Authentication to verify the user. This means that user credentials and sensitive information are protected during application use. Additionally, we should implement end-to-end encryption and secure API access so that data isn't only secured on the backend.