

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 | 1/25/2025 | Richard Adler | Software design document for an environment for a web-based management system for Draw It or Lose It for client Gaming Room |

**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)

Gaming Room is looking to expand the scope of their current game, Draw It or Lose It. The game consists of teams guessing an image as several clues are rendered over the course of a minute. It is currently only available on Android. Gaming Room is looking to expand to multiple platforms and seeks to create a web-based version of Draw It or Lose It and needs an environment to host the web-based version. A Java-based hosting service will be created to facilitate this task.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

* Only one instance of each game may be created
* Team and player names must be unique
* The application must run on multiple platforms
* Teams must host multiple players
* Games must host multiple teams

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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 is renowned for its security, is built to last, and offers support for the most popular web-based server software systems. However, Macs carry a higher up-front cost, expensive software licensing, and have limited software available to them. The inflexibility of Macs is a trade-off for excellent security, the high up-front costs will likely pay off later in the life of the servers. | Linux distributions do not have licensing fees, boasts high customizability and adaptability due to the open-source OS, offers a breadth of software options, hosts restrictive securities and permissions, and can be scaled relatively easily. Linux is more complex, needing to be tailored specifically to hosting the game, and offers limited support for some enterprise applications. | Windows offers Windows Defender and firewalls for security, an intuitive interface that most are accustomed to, carries support for most software, is designed to scale easily, and is sustainable, needing less maintenance than other options. Windows also comes with high licensing fees, holds many complex customizations and options, and has less security overall than some other platforms. | Mobile devices are not recommended for use as servers. They have serious flaws with uptime, processing power, server security, and system maintenance as they are not designed for this use case. |
| **Client Side** | Compatibility for multiple browsers is a must, ensuring the application runs smoothly and is responsive across various web browsers. Additionally, the game must adhere to Apple’s accessibility guidelines. | The application must be designed to format correctly for various popular browsers. Integration tests and feedback from game testers will need to be implemented to ensure the game runs correctly. | Multiple browsers will need to be accounted for when developing the game. Consistency in formatting and performance for common web browsers is a must. This will likely require multiple tests to ensure compatability. | The game would need to be designed for a mobile-oriented approach. Client-side software for mobile needs to account for the touch screen and lack of keyboard, alongside formatting for various mobile devices. The game also needs to be formatted for both IOS and Android devices and expect many different models of Android. |
| **Development Tools** | Macs have access to xCode, SwiftUI, SwiftPlayground, and AppKit for IDE’s and app builders. The game could be built in Swift, Python, Java JavaScript, and PHP ideal languages for web-based features. | Linux supports most IDE’s, including Eclipse, PyCharm, Visual Studio Code, and many others. Languages such as Java, JavaScript, PHP, and Python are well suited to hosting a web-based application. | Windows hosts Visual Studio and Visual Studio Code alongside many language specific IDE’s such as PyCharm and Eclipse. Java, JavaScript, PHP, and Python contain many features to manage the game via the web. | Java, using Android Studio is preferred for Android. Swift, written in xCode is preferred for IOS. Mobile devices are not well suited to development and is strongly recommended to use a more standard environment. |

## 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**: The Gaming Room should utilize a Linux distribution running on cloud-based servers. Cloud-based services operate on a pay-by-use basis, and this model would not limit The Gaming Room during expansion to its current hardware or force an investment of additional servers to maintain the game experience for a growing player base. This does place trust in the third-party service for integrity of data and maintenance, but this tradeoff is worthwhile as Draw it or Lose it expands. Amazon Web Services (AWS) is highly recommended and utilized by many serverless architectures and would make an ideal choice for The Gaming Room to host Draw it or Lose it. The service is commonly used and features a user-friendly interface for those unfamiliar with it, supports scaling and offers a pay-per-use model to ensure that The Gaming Room is able to expand without issues. AWS also hosts a robust security suite and frequently adopts new technologies as they arise, so that The Gaming Room can stay current and spend resources on expansion rather than working through in-house security and storage issues.
2. **Operating Systems Architectures**: One of Linux’s biggest strengths is its open-source nature. This has two major advantages as compared to other operating systems, mainly that there are no licensing fees for many distributions, and it is designed to be flexible and customizable, easily adapted to a myriad of needs. Linux servers are renowned for their stability, leading to longer uptime and less maintenance overall. They also fit well into a modular system, so that as Draw it or Lose it grows, architecture can be integrated into the existing system seamlessly. Commonly used benchmarking tools such as perf are supported to ensure system performance and identify bottlenecks, gaining insight into CPU usage and cache misses. Debian is recommended for the Linux distribution considering its ease of use, consistent updates, and high degree of customizability while offering an approachable and easy to manage user-interface. Debian is also recommended for its overall stability, as updates are carefully managed, and servers typically do not have concerns surrounding updates to ancillary applications. This combined with out of the box security features make Debian an ideal choice for The Gaming Room’s servers.
3. **Storage Management**: Linux utilizes a logical volume manager (LVM) to manage storage throughout disk space. LVM allows for dynamic allocation of storage, providing a flexible solution for managing data storage. This is accomplished through a three-step process, physical volume, volume groups, and logical volume. All storage, whether virtual or physical, is treated as physical storage, and is added to the available pool of storage resources. Volume groups are responsible for hosting buckets of physical storage, storing various like resources together across these buckets. Grouping is an abstraction of the physical volume devices, bearing in mind that all storage is treated as physical throughout LVM. Logical volumes function similarly to system partitions, abstraction culminates into an integrable sect of storage that interacts with the operating system seamlessly. Linux’s ability to scale with a modular approach will allow The Gaming Room to expand without a more complex plan or lengthy service interruptions.
4. **Memory Management**: Linux features many automated and flexible solutions for memory management out of the box. The Out of memory (OOM) killer is a tool automatically employed by the OS to eliminate process when, and only when the system is critically low on memory. While the process is automated, the OOM killer checks the OOM score of a process, favoring processes with a lower score and keeping processes with a high score running. This OOM score can be customized for each PID generated, ensuring that critical processes are not subject to end unexpectedly via OOM killer. Transparent huge pages (THP) allow for simplified memory management from a user perspective by allocating up to 1 GB pages of memory to reduce the amount of translation lookaside buffer (TLB) misses. TLB is the process of translating top level addresses to usable registers; by using huge pages to address large blocks of memory at once, significant workload is saved on the TLB and thus improves performance.
5. **Distributed Systems and Networks**: Load balancing is essential to ensuring smooth performance, especially during high demand periods. Distributing workloads across the available system resources by first running requests through a load balancer to meter tasks to individual servers allows those servers to operate efficiently. Another benefit to employing AWS services is their cloud load balancer, further reducing the workload on The Gaming Room to expand Draw it or Lose it as fast as possible. Role based access control (RBAC) should be used so that administrators can have access to the tools they need, and users are limited only to the aspects that are necessary for them to experience the game. RBAC ensures that all users have access to the resources provided in their role across all servers in the system.
6. **Security**: AWS comes standard with many security features. AWS Security Hub gives an easy-to-understand overview for administrators on any security risks that are detected. Cloudtrail monitors back-end activity, logging any API calls and identity actions, while Guardduty monitors client-side accounts, continually scanning for malicious activity. AWS offers tools for encryption, further protecting data in the event of a breach. The cloud service is also set up for the principle of least privilege, giving users only the permissions they need for their specific role. Furthering the principle of least privilege, RBAC combined with two-factor authentication will ensure that users have access only to the resources they should and help prevent unauthorized logins. In further pursuit of this, a readily accessible page detailing The Gaming Room’s contact email, communication protocols, and clear language that company representatives do not ask for two factor auth codes will be constructed to further reduce unauthorized logins.