

<Name-of-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 | 06/06/2025 | Shariona Johnson | Project 2 Completion |

**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 currently has Draw It or Lose It available only on Android, but they want to reach a wider audience by making the game accessible on multiple platforms, including desktop systems (Mac, Linux, Windows) and mobile devices (iOS and Android). As someone who appreciates user-friendly design and cross-platform compatibility, I think it's exciting to help design a flexible, responsive, and secure web-based version of the game. This software design document outlines the requirements, platform comparisons, and my personal recommendations based on usability, cost, performance, and development needs. This will help The Gaming Room make a smart decision about expanding their game.

## Requirements

The client’s business and technical requirements include:

* Expand the game to a broader audience by moving from Android-only to a platform-independent web-based model.
* Ensure the new version is responsive, meaning it works across various screen sizes and device types.
* Use a secure client-server model to handle game logic and real-time interaction.
* Support thousands of users at once with minimal lag.

Implement good security practices to protect user data

## [Design Constraints](#_2et92p0)

Since this game will be web-based, it has to function reliably in a distributed system where the backend server handles all the logic and clients access it through browsers. That means the server must be strong enough to support multiple users at once, and the front end must be lightweight, responsive, and browser-compatible. Also, because the game must work on mobile and desktop, design consistency and efficient communication between clients and the server are critical. I also had to consider that the budget may be limited, so I leaned toward open-source and cost-effective tools when making decisions.

## [System Architecture View](#_ilbxbyevv6b6)

This game will be structured as a traditional client-server model. The server side will be deployed using a Linux-based cloud platform to manage game state, user data, and image rendering. The client side will be developed using web technologies like HTML, CSS, JavaScript, and frameworks such as React, which allows us to support mobile and desktop platforms through a single codebase.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram includes several key components: Game, Team, Player, and GameService. GameService follows the Singleton pattern so only one instance exists in memory, which helps keep things consistent across different users and sessions. Teams and Players are managed through the Game class, and the Iterator pattern helps avoid duplicate entries. This structure keeps things organized and efficient, following object-oriented principles like encapsulation and reusability. It’s especially helpful for scaling the game later on.

**"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** | macOS isn’t typically used for web servers, so while it’s possible, it’s not ideal. Tools are more limited and Apple hardware is expensive. | Linux is perfect for server-side hosting. It’s fast, secure, and free. It’s the standard for most web apps and is highly reliable for scaling. | Windows Server works but comes with licensing costs and more complex setup. It’s also more prone to viruses, so extra protection is needed. | Mobile devices aren’t meant to be servers. They’re good for users but not for hosting the actual game backend. |
| **Client Side** | macOS users will access the game via Safari or Chrome. Development is smooth, but Apple’s approval process for apps (if needed) is strict. | Works well through Firefox or Chrome on Linux. Not as common for gamers but still important for inclusion. Open-source and easy to test. | Windows is widely used. Chrome, Firefox, and Edge will all need to be tested. Users are familiar with gaming on this OS. | This is where we expand the most. Supporting both Android and iOS will take extra work but pays off. Using web views and responsive design helps keep things simple and unified. |
| **Development Tools** | Xcode, IntelliJ, or VS Code work well on Mac. Tools are free, but the hardware costs more. | Linux supports a bunch of open-source IDEs like VS Code, Eclipse, and NetBeans. Easy for backend and full-stack development. | Windows supports Visual Studio, which is powerful but sometimes expensive. Also supports Java, Eclipse, and VS Code. | We can use Android Studio and Xcode for mobile, or go cross-platform with tools like Flutter or React Native to save time and avoid separate codebases. |

## Recommendations

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

**Operating Platform**: For expanding Draw It or Lose It across multiple computing environments, I recommend using **Linux-based servers**—specifically **Ubuntu Server LTS (Long-Term Support)** as the main server operating platform. Linux provides high flexibility, strong security, and excellent performance, all while being cost-effective due to its open-source nature. Ubuntu LTS versions offer reliable stability, long-term updates, and strong community and enterprise support, which will help maintain a smooth experience as the game scales across platforms.

**Operating Systems Architectures**: Linux follows a monolithic kernel architecture, meaning the core system services—like memory, file system, and process management—run in kernel space for maximum performance. This architecture is especially beneficial for gaming platforms like ours that need fast, efficient responses between system components. The modularity of the Linux kernel also means I can customize or add features as needed without bloating the entire system.

For client-side environments, *Draw It or Lose It* will run on multiple platforms, including Windows (NT architecture), macOS (hybrid XNU kernel), and Android (modified Linux kernel). Each of these uses a slightly different model, but the core functions like memory, file, and user management operate effectively through abstracted APIs. The back-end Ubuntu server can easily support connections from these various front-end environments thanks to standard protocols like HTTP/HTTPS and WebSockets.

**Storage Management**: For the storage management system, I recommend using ext4 (FourthExtended File System) on the Linux server. It’s reliable, supports large files and directories, and includes journaling, which helps recover from crashes. To manage the large image library of 200+ HD image files (each 8MB), I would also set up Amazon S3 or another cloud-based object storage as a CDN (Content Delivery Network). This allows the images to be quickly accessed by clients without burdening the server with file-serving overhead.

Additionally, a PostgreSQL database will be used to manage user accounts, game data, and metadata related to image usage. This gives us a powerful, relational structure with strong performance and ACID compliance.

**Memory Management**: Linux uses a virtual memory management system that includes paging, demand loading, and swapping. For *Draw It or Lose It*, this means the server can allocate memory to active processes dynamically while managing background data efficiently. The Linux kernel also uses a Least Recently Used (LRU) page replacement algorithm, which helps prioritize memory use for the most frequently accessed processes—perfect for game services where performance needs to be consistent.

If memory use spikes, the system will use swap space intelligently, ensuring that the most important operations (like game logic and image delivery) are prioritized. On the client side, platform-specific garbage collection (like Android’s ART or Windows’ .NET GC) will ensure smooth gameplay without memory leaks.

**Distributed Systems and Networks:** To make *Draw It or Lose It* work across different platforms, we’ll use a RESTful API on the server, built with a web framework like Dropwizard or Spring Boot. This keeps communication between server and clients standardized, regardless of the front-end platform.

The application will run as a distributed system, meaning the client apps (on web, iOS, Android, etc.) will communicate with the central server via a secure network. The network will use HTTPS with TLS encryption to secure data and support fallback or retry mechanisms in case of connectivity issues.

For dependencies, services will be loosely coupled. The image-rendering engine, game logic, and user authentication services will operate independently using microservices or service-oriented architecture, allowing more resilience and easier updates. If one service fails (e.g., image rendering), it won’t take the whole system down.

**Security:** Security is crucial, especially since users will be interacting from many devices and platforms. On the server, I’ll implement role-based access control (RBAC) to restrict administrative functions and secure APIs with OAuth2 authentication tokens. All data transferred will be TLS encrypted to prevent interception.

Each user’s sensitive information (like login credentials) will be hashed using bcrypt before storing it in the database. Additionally, any API call from the client must pass through an authentication middleware that checks for valid tokens before executing protected actions.

On the network level, a firewall (like UFW) will only allow necessary ports (HTTPS/443, SSH/22 for admin access), and all unused services will be disabled. On top of that, we’ll use logging and audit trails to monitor suspicious behavior and make sure security policies are being followed.