

<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 | 10/25/2025 | Cha Yang | Completed evaluation of all platforms and recommendations for The 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)

The Gaming Room wants to take its Android-only game, Draw It or Lose It, and make it work everywhere—on phones, tablets, and computers. To do this, we’ll turn the app into a web-based game that runs inside a browser. The server will manage players, teams, and scores, while the client will show the drawings and timers. This setup keeps things simple for users (no installs needed) and allows thousands of players to join at once. The best way forward is to host the backend on Linux and use modern web tools to build the frontend so that it’s fast and flexible.

## Requirements

Business requirements:  
- Let players use any device without downloading an app.  
- Handle thousands of simultaneous games.  
- Keep costs low and performance high.  
  
Technical requirements:  
- Build with HTML, CSS, and JavaScript so it works across browsers.  
- Use a Linux-based web server.  
- Store data in a secure database like PostgreSQL.  
- Ensure real-time gameplay using WebSockets or similar technology.

## [Design Constraints](#_2et92p0)

The main challenge is building a system that runs smoothly in a web-based distributed environment. We have to make sure that the game updates instantly, no matter what device players are using. Other constraints include internet latency, security, and compatibility across devices. These affect development by requiring lightweight code, strong backend servers, and efficient communication systems.

## [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)

The UML class diagram shows how objects like Game, Player, and Team connect. Each player belongs to a team, and each team participates in a game. The Game class controls rounds, timing, and scoring. We use OOP principles like inheritance and encapsulation to keep code organized and reusable. This makes it easier to manage player data and add new features later without breaking existing code.

**"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 can host web servers but is rarely used this way. It's stable but limited by Apple’s licensing. Not ideal for large-scale hosting because cloud providers favor Linux. | Linux is the best option for hosting. It's open-source, reliable, secure, and runs well with technologies like Node.js, Nginx, and PostgreSQL. There are no licensing costs, and scaling is easy. | Windows Server can host applications well, especially if built with Microsoft tools. However, licensing fees and extra configuration make it more expensive. It’s less efficient than Linux for lightweight web apps. | Mobile devices are not designed to host servers. They connect to the main game server through browsers. We focus on optimizing performance and battery life when accessing the web-based game. |
| **Client Side** | Mac browsers (Safari, Chrome, Firefox) support the web app easily. Testing time is moderate, but development costs are shared with Linux and Windows since the code is browser-based. | Linux users access the same web app through Chrome or Firefox. Development is simple and costs stay low, though testing on multiple distros is wise. | Windows is the most common desktop platform. Testing on Edge and Chrome ensures wide compatibility. Performance is strong and development tools are flexible. | Mobile browsers (Safari, Chrome) require responsive design for small screens and touch controls. Extra testing ensures smooth play even on slower networks. |
| **Development Tools** | Languages: JavaScript/TypeScript for frontend, Node.js or Python for backend. Tools like VS Code and Git are free and easy to use on Mac. | Linux supports all major development tools and languages. Great for running servers and CI/CD pipelines. Everything is open-source and cost-effective. | Windows supports the same stack and adds tools like Visual Studio. Developers can use WSL (Windows Subsystem for Linux) to match Linux environments. | Developers use browser dev tools, Android Studio, or Xcode if they package the game into an app later. Most tools are free; extra cost comes from testing on multiple devices. |

## 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 the best platform for hosting the Draw It or Lose It backend. It's secure, stable, and low-cost. Most modern web apps run on Linux servers, so support and scalability are excellent.
2. **Operating Systems Architectures:** We’ll use a Linux-based x86\_64 or ARM64 architecture with containers for portability. This lets us scale easily and run on cloud providers like AWS, Azure, or Google Cloud.
3. **Storage Management:** PostgreSQL will store player info and game results, while S3 or similar storage will hold drawings and images. This combination keeps data reliable, secure, and easy to back up.
4. **Memory Management:** Linux handles memory efficiently through virtual memory management and caching. It uses memory only as needed, freeing it for other processes automatically. This ensures smooth performance even with thousands of players.
5. **Distributed Systems and Networks:** We’ll use REST APIs for general communication and WebSockets for real-time updates. This allows the game to run smoothly across devices. If a connection drops, the player can rejoin instantly without losing data.
6. **Security:** We’ll protect users with HTTPS encryption, strong authentication, and password hashing. Player data will be encrypted and stored safely. The Linux firewall and SSL certificates will keep everything secure.