

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 2.0 | 06/16/2025 | Chris Prempeh | Final submission with Project 3 Recommendations completed |

**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 expand their Android game, Draw It or Lose It, into a web-based version that works across platforms. As a Technology Consultant at CTS, my job is to guide that transition by designing an efficient, scalable software solution.  
  
This design outlines a system that allows multiple teams and players while keeping the game instance unique using a Singleton pattern. It ensures game and team names remain unique and sets the stage for a smooth development process, even as hardware needs evolve.  
  
The focus is on flexibility, maintainability, and keeping the structure clean so future updates, like adding multiplayer support or user profiles, can be built without overhauling the core system.

## Requirements

- The game must support one or more teams, each with multiple players.  
- Game and team names must be unique.  
- Only one instance of the game should exist in memory (Singleton).  
- The app will be web-based and needs to support multiple platforms.

## [Design Constraints](#_2et92p0)

Since this game is moving to a web-based, distributed environment, there are a few important limitations to keep in mind. The app needs to run consistently across different browsers and devices, which means we can’t rely on platform-specific features. Everything must be responsive and lightweight enough to handle mobile and desktop users smoothly. Another constraint is real-time performance. Multiplayer games need low latency and fast data sync between players, so the system must be optimized for quick communication and minimal lag. On top of that, security and scalability need to be built in from the start, since more users will likely join over time.

## [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 diagram includes:  
  
- Classes: Game, Team, Player  
- Associations:  
 - Game → has many → Teams  
 - Team → has many → Players  
  
OOP Principles Used:  
- Encapsulation: All class data is private with public getters/setters.  
- Inheritance (if extended later): Future roles like Moderator or Viewer could extend from a User class.  
- Abstraction: The logic behind adding/removing teams or players is hidden within class methods.  
- Polymorphism: Useful if roles or player behaviors expand in the future.  
  
This setup satisfies all client requirements while leaving room for future upgrades.

**"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** | While macOS can technically be used to host web applications using servers like Apache or Nginx, it’s not commonly used in production environments. Hosting at scale on macOS is rare due to hardware limitations, limited support in enterprise environments, and the need to purchase Mac hardware, which increases cost. It’s more commonly used for local development than deployment. | Linux is the most widely used operating system for web-based application hosting. It supports popular server software such as Apache, Nginx, and Tomcat, and is compatible with most cloud platforms (AWS, Azure, GCP). It’s open-source, which means no licensing fees, making it a cost-effective and scalable solution for hosting web applications. | Windows supports web hosting using Internet Information Services (IIS) and is particularly suited for .NET-based applications. While it offers solid performance and integration with Microsoft tools, it comes with higher licensing costs and typically requires more resources to manage compared to Linux. It’s a reliable but more expensive choice for server deployment. | Mobile platforms (iOS and Android) are not used to host web applications. Instead, they serve as client devices that connect to the web server. As such, they are not suitable for server-side deployment and have no relevance in hosting web apps. |
| **Client Side** | To support Mac users, the app just needs to run smoothly in browsers like Safari, Chrome, and Firefox. Since this is a web app, making sure the design is responsive using HTML, CSS, and JavaScript is the priority. Only downside is that testing on Safari might require a Mac. Time and cost are reasonable, but having devs who know the Apple environment helps. | Linux users mainly use Firefox or Chrome. If the web app works across modern browsers, you’re good. The main thing is testing it on different Linux versions to catch any weird bugs. Development is straightforward and doesn’t add much cost, but it’s still worth covering all the bases. | This is where most users will be, so everything needs to work smoothly in Chrome, Edge, and Firefox. The web-based setup makes things easier since we don’t need to build a separate Windows app. It’s easy to test and pretty affordable in terms of time and cost. | The goal here is making sure the app looks good and works well on phones and tablets. It needs to adjust to different screen sizes and touch input on both Android and iOS. No need to build separate mobile apps, but you’ll want solid mobile web dev skills and a good amount of testing on real devices to be safe. |
| **Development Tools** | Macs are solid for frontend development, especially if you’re using tools like Xcode for iOS, VS Code, or Android Studio. Most web development tools work fine on Mac. The only catch is that Xcode is only available on Mac, so if you’re building or testing for iOS, you’ll need Apple hardware. No extra licensing costs unless you’re submitting to the App Store. | Linux supports most major development tools like VS Code, Eclipse, and IntelliJ. It’s a strong option for backend development, especially when working with open-source tools and servers. It’s free, stable, and popular with developers. No licensing issues, and it’s easy to set up. | Windows gives you access to tools like Visual Studio, VS Code, and Android Studio. It works well for both frontend and backend development. You might run into a few setup quirks depending on the stack, but nothing too major. Some versions of Visual Studio have licensing fees, but there’s a free Community Edition that covers most needs. | You won’t be coding directly on mobile devices, but you will need tools like Android Studio and Xcode to test how the app runs in mobile browsers. Chrome DevTools, Safari Web Inspector, and mobile emulators will help a lot with this. Android tools are free. For iOS testing, a Mac is required, which could add some cost if your team doesn’t already use one.do I |

## 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**: I’d recommend going with Linux to run the game. It’s reliable, free to use, and works great with a lot of different tools and frameworks. It’s also fast, secure, and handles high traffic well, which is exactly what this kind of game needs.
2. **Operating Systems Architectures**: Linux has a clean, modular setup, which makes it easier to manage and tweak depending on what you need. It can handle multiple users and tasks at once, and it keeps everything isolated so one problem doesn’t mess up the whole system.
3. **Storage Management**: Using cloud storage like AWS or Azure would be the best move for holding game data, with something like MySQL or PostgreSQL to handle the team and player info. Linux connects easily to both local and cloud storage and adds extra protection with its file system.
4. **Memory Management**: Linux manages memory well. It uses virtual memory so it can keep things running even when the physical memory is low. It also gives good control over background processes, which is helpful for a multiplayer game that’s always running and dealing with a lot of data.
5. **Distributed Systems and Networks**: Since the game will run across different platforms, we’ll need to split things up using a distributed setup. That means the game logic, website, and database can each run on their own. Linux works great with tools like Docker and Kubernetes, which help keep everything connected and running smoothly.
6. **Security**: Security is huge for a game like this. Linux has built-in tools that help protect the server, like firewalls and user permission settings. I’d also recommend using HTTPS, login systems like OAuth, and encrypting all the sensitive data. Keeping things updated and locked down will go a long way in keeping user info safe.