

The Gaming Room

# **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 | 01/24/2025  02/09/2025  02/22/2025 | Savannah Mattoon | -Overseeing design restraints and requirements  -Evaluating characteristics, advantages, and weaknesses of various platforms  -Reccomendations |

**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 would like to develop a web-based game that serves multiple platforms based on their current game, *Draw It or Lose It*. This game is currently only available in an Android app only. The goal is to expand availability to different platforms while maintaining the integrity of the game.

## Requirements

* Supported by multiple platforms
* Expand availability
* Each team must have unique name
* One instance of game at a time can exist
* Must support multiple teams
* Must be responsive in real-time.

## [Design Constraints](#_2et92p0)

The design for the software should be supported on different platforms and have flexibility enough to adapt to different technological mediums. The game should be able to adapt to fit a growing customer base and keep up with demand. The game play should be responsive in real-time to assure a smooth game play environment for customers. One instance of a game should exist at one time and multiple teams should be able to play with each team having unique identifiers.

## [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 shows how the program will interact. The Entity class is the base class to the Game, Team, and Player classes. It allows these classes access to its attributes through getter methods demonstrating encapsulation. The Game class contains a list of Team objects that also manages Player objects demonstrating composition. The GameService class uses a singleton pattern and only allows one instance of itself to be used, and each game and team must have unique identifiers.

**"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** | Pros:  -compatible with many open-sourced tools and server environments  -offers strong processing power  -developer friendly  -offers strong security    Cons:  -not good for continuous, high-traffic hosting  -high cost for hosting  -lacking in server management tools  -not optimized for resource intensive processes | Pros:  -open source  -easily scalable to handle traffic  -highly efficient  -stabile and reliable  -strong built-in security  -software support  -extensive online resources  -cost effective  Cons:  -steep learning curve  -certain tools may not be available for Linux  -may not be compatible with some hardware | Pros:  -User friendly  -wide software compatibility  -seamless integration with Microsoft tools  -robust support such as updates and patches  -familiar environment  Cons:  Licensing costs can be expensive  -less performance efficiency due to being more resource-heavy  -prone to security breeches  -limited open-source support  -not as efficiently scalable | Pros:  -Portability  -user-friendly touch-based interfaces  -native integration  -large user audience  -easy cloud integration  Cons:  -limited resources  -limited performance  -hosting applications drains battery  -security risk if device is stolen  -limited scalability  -can have connectivity issues |
| **Client Side** | Cost:  -expensive hardware costs  -may require licensing  -increased cost for cloud service if scalability is required  Time:  -cross-platform testing  -development time to ensure app works on different screen types  -optimization  Expertise:  -knowledge of web technologies to ensure combability across browsers  -server/hosting  -familiarity with macOS development tools for Apple integrations | Cost:  -free open source  -may cost more for scalability  -may require paid licenses for enterprise-level tools  Time:  -cross-platform testing  --optimization  -setup configurations  Expertise:  -Linux knowledge  -knowledge of web technologies to ensure combability across browsers  -performance tuning  -familiarity with development | Cost:  -licensing fees  -cloud hosting may cost more than Linux  -development tools may require license  Time:  -cross-platform testing  -development time to ensure app works on different screen types  -setup configurations  Expertise:  -knowledge of windows servers  -server-side development  -cross-browser development  -security management | Cost:  -development tools  -app store fee for membership  -testing across devices  -cloud services  Time:  -cross-platform testing  -development time to ensure app works on different screen types  -approval from app store  Expertise:  -Mobile development  -UI design  -performance optimization  -backend integration  -mobile security |
| **Development Tools** | Languages:  -JavaScript  -HTML5/CSS3  -Python  -Node.js  Tools/IDE  -Xcode  -visual studio  -Git | Languages:  -JavaScript  -HTML5/CSS3  -Python  -PHP  -Node.js  Tools/IDE  -Docker  -Git  -Atom  -visual studio code | Languages:  -JavaScript  -HTML5/CSS3  -Python  -c#  -Node.js  Tools/IDE  -Docker  -Git  -IIS  -visual studio code  -visual studio | Languages:  -JavaScript  -Swift  -Kotlin  -Dart  -c#  Tools/IDE  -Xcode  -Android studio  -visual studio code  -Expo  -Git |

## 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**: My recommendation is that The Gaming Room go with **Windows** for both server and desktop environments while also supporting mobile devices for broader access. Windows has a large user base and great development tools making it easier to reach more players and create engaging applications. Mobile gaming is popular right now so targeting both iOS and Android will boost user engagement. While there are some challenges like licensing costs and device fragmentation this approach strikes a good balance between performance and accessibility.
2. **Operating Systems Architectures**: Architecture for Windows and mobile devices is efficient and user-friendly. Windows provides fast process management and strong memory capabilities along with secure user and kernel modes. It also has useful APIs like WinAPI and .NET for development. Both iOS and Android use layered architectures that separate core functions from applications making updates easier and boosting performance. Android is based on the Linux kernel while iOS uses the XNU kernel with development tools like Android Studio and Xcode. This setup helps create engaging experiences for "Draw It or Lose It" across all platforms.
3. **Storage Management**: For storage management I recommend using cloud services. These options offer scalable, secure, and reliable storage that can handle growing data needs from user profiles to game content. Cloud storage also ensures easy access across multiple platforms whether on desktop or mobile. With built-in encryption and high availability these services provide the security and flexibility needed to support "Draw It or Lose It."
4. **Memory Management**: Windows and mobile devices all have solid memory management to keep **"Draw It or Lose It"** running smoothly. Windows uses virtual memory and dynamic allocation to manage resources efficiently while also protecting memory between processes. On mobile, Android handles memory with garbage collection to avoid leaks and iOS uses memory optimization techniques. Both platforms also manage background apps to free up memory for active tasks promising smooth gameplay.
5. **Distributed Systems and Networks**: To make **"Draw It or Lose It"** work across different platforms the game can use a **client-server system** where the server manages the game data and the client communicates with it through APIs or WebSockets for real-time updates. The game can handle network issues with load balancing, caching, and offline modes. WebSockets provide instant updates across platforms while HTTP/HTTPS manages basic communication. This setup ensures smooth gameplay even with slow or unstable connections.
6. **Security**: To keep user info safe in **"Draw It or Lose It"** **encryption will be used** for data and secure logins with **Multi-Factor Authentication.** We'll also make sure the game follows **secure coding** to avoid vulnerabilities.