

<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 | 10/10/10 | Karlo huerta | Adding to what project one was. |
| 2.0 | 10/26/25 | Karlo huerta | Completing the UML Diagram Description |
| 3.0 | 10/26/25 | Karlo huerta | Recommendation complete. |

**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 and grow into other platforms like desktop and mobile browsers. The game Draw it or Lose with the recommendations will work seamlessly from a cloud environment server. The new system will work off the based code and continue to use object oriented components from the UML diagram.

## Requirements

* *Expand from android*
* *Allow thousand of players to connect.*
* *Have consistency between mobile and desktop browser.*
* *Scalability*

*Technical*

* *Make sure responsive layout for both desktop and mobile*
* *Token based authentication works.*
* *Use API to work correctly*

## [Design Constraints](#_2et92p0)

* Token based authentication needs to keep data secure.
* Must work across the board between applications
* Backend must handle thousands of users.

## [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)

Game, Team, and Player all show inheritance from entity allowing every domain object to have a unique ID and Name without repeating the code. This shows within Game there will contain Teams(0..\*) as well as in a team will contain many Players (0..\*) and GameService will contain many Games (0..\*). GameService class is implemented as a Singleton to make sure theres one central manager that creates and track all games, teams, and players while also assigning unique ID’s.

**"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 small-scale applications but isn’t practical for large production environments due to licensing and hardware limitations. | Linux is secure , open source, scalable and used in all major cloud platforms. | Works best for enterprise setups using .NET but cost more to maintain | Cannot work as web servers . they lack stability and security. They can host small demos. They have limited connectivity and uptime . they have no licensing cost. |
| **Client Side** | Cost will be medium – you need apple hardeware and MAC OS but no licensing fee. Time is low design could be made fast and responsive and easy to test. Devs needs experience testing in safari and make sure IOS compatible | Cost is low free tools like visual studios , VS code, docker, git  Time is low web apps perform well through chrome and firefox. | Cost is medium due to windows licensing and testing tools like visual studios adds cost. Time is low to medium. Devs will need to understand browser variations and window firewalls. | Cost is medium for mobile testing tools across the multipls platforms. Time is high due to testing for screen size and data optimization and performance |
| **Development Tools** | Uses Xcode, VS Code, and Docker for development and front-end testing | Uses VS code, docker, git, node.js best for backend environments, works great with cloud services. | Visual studio, vs code, docker, .NET | Android studios, xcode, chrome devtools |

## 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 recommend running the application through a Linux based server environment using AWS, amazon web services. Linux is commonly used for cloud environments because its lightweight, stable, open sourced and has a lot of development tools. AWS allows flexibility for resources and the application will be easily scalable since it is done automatically based on how many players are online by AWS.
2. **Operating Systems Architectures**: Linux uses a modular operating system where the kernel manages functions like memory, process scheduling, and device drivers. The applications will run in a user space and will communicate through system calls. This will allow the Gaming Room to be secure and stabile consistently.
3. **Storage Management**: AWS supports relational databases and object storage systems. This allows for us to store scoring data and large image files through multiple data centers for reliability and durability. While linux will be the infrastructure and combining the local file structure with AWS’s scalability of storage services this will allow for quick access and retrieval of the images during gameplay so no data loss happens.
4. **Memory Management**: Linux uses virtual memory management that uses secondary storage if there isn’t enough physical RAM to use and is done through paging, segmentation and caching. This will allow the server to run smoothly when multiple users are online and connected.
5. **Distributed Systems and Networks**: since we are trying to run as part of a distributed system, AWS will allow that by using standard protocols like http and https to communicate through the internet. when a player performs an action and submits it. That info will be sent to a central game server and show it to all players in that session.
6. **Security**: <since the server will be running on Linux, the system benefits from several security features that are backed by AWS. Linux supports a permission-based access model, where files and system resources can be restricted at the user, group, and administrator (root) levels (Silberschatz, Galvin, & Gagne, 2008). This prevents unauthorized access to core game services and user data.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2008). *Operating system concepts* (8th ed.). Hoboken, NJ: John Wiley & Sons.