

<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 |
| --- | --- | --- | --- |
| 3.0 | <06/22/2025> | Jasmine Villarreal | Last version of the project |

**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 purpose of this document is to outline the design plan for developing a web-based game for our client, *The Gaming Room*. The client aims to create a game inspired by the popular TV show “Win, Lose, or Draw”, and intends to name the application “Draw It, or Lose It.” The objective is to develop a cross-platform game that is both user-friendly and accessible on a variety of devices.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* Web based support: The game application needs to be developed for a web based platform. The plan is to be able to be compatible with most devices.
* There has to be unique names so there are no name conflicts.
* Single instance is also important to this project.

## [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 purpose of this report is to analyze the UML diagram provided for The Gaming Room application. We can see the relationship between classes and the OOP principles apply to help meet the system design.

The UML has a couple of key classes with the Entitiy.java being the superclass. The subclasses are Game, team and Player. With game service as the Singleton class and program Driver as the entry point.

The Entity class serves as the superclass with shared attributes “id” and “name”. the Game Class will inherit from “Entity” and will contain a list of Team Objects. It helps to identify a specific game and to be able to “add teams”. The Team class will also inherit from the “Entity” to help maintain a player list. The “Player” class will also inherit form the entity and represent individual players in teams. Game Service will help demonstrate the singleton design so there is only one instance of a game at a time.

**"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** | Mac can host web applications but is less commonly used for server environments. It's stable and secure. It’s better to use for development because of the limited scalability. | Linux is the most popular choice for hosting web-based applications. It’s open-source, really customizable, and secure. It supports most server technologies and is cost-effective, but may come with a learning curve. | Windows servers offer good support for Microsoft technologies and has a user-friendly development ecosystem. But it does have a issue with security vulnerabilities. | Mobile devices are not suitable for hosting web apps due to limited computing power, OS restrictions, and battery constraints. They're meant for client-side use, not server hosting. |
| **Client Side** | Supporting Mac clients requires ensuring compatibility with Safari and macOS UI standards. Development tools are available, but testing on actual Mac devices may be needed. It's less common than Windows in general user bases. | Linux client support is important for technical users. Developers need to ensure cross-browser and OS compatibility. Testing on Linux is essential, though its user base is smaller than Mac or Windows. | Most users use Windows, so client support is essential. Compatibility with major browsers like Chrome and Edge is important. Development and testing on Windows are generally straightforward. | Mobile support is essential. Applications must be responsive and touch-friendly. Testing on both iOS and Android is needed, which adds development and QA time. Frameworks like React Native or Flutter help reduce effort. |
| **Development Tools** | Xcode, VS Code, and Terminal are common tools. Mac supports development in Swift, Python, JavaScript, and more. Useful for iOS development and general web dev. | Popular tools include VS Code, Eclipse, and the Linux terminal. Great support for Java, Python, and server-side languages. Ideal for back-end and full-stack development. | Common tools are Visual Studio, VS Code, and PowerShell. Strong support for .NET, C#, and front-end development. User-friendly but may require more resources. | Tools like Android Studio (for Android) and Xcode (for iOS) are used. |

## 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** This will be a web-Based platform to allow the game to be more accessible to everyone. Web Based platforms offer an easier way to distribute the application and for easier updates. This will create a larger user platform.
2. **Operating Systems Architectures**: we should consider a multi-tier architecture to help ensure a broad compatibility. This will ensure the usage of web technologies such aqs css, html , and javascript. These tools are also well known to be cross platform friendly.
3. **Storage Management**: We should consider a combination of a relational database and cloud storage. Cloud storage offers a strong integration with web applications. As for structured games and user data a rational database (such as MYSQL or AWS RDS) will ensure data integrity and security .
4. **Memory Management**: We should consider modern memory management such as garbage collection or memory pooling for repeated game usage.
5. **Distributed Systems and Networks**: to be able to support cross-game play we should consider a distributed web application with the possibility of APIs to ensure communication between the client and the server. This will help insure that the servers can manage multi games and handling communication between players on various devices.
6. **Security**: Since this will be a multi device platform we need to have security measures that will protect our users. We can consider the basics of web based design such as email, encrypted password, and a unique username/player name. from a application based approach we can consider HTTP for encrypted data transmission and regular security audits.