

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 | 01/21/23 | Trevor Mickelson | This is the first commit |

**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)

Hello, the software requests for your application will not be a problem to implement. It is worth noting that a database will be required for any kind of persistence in the application. And if this game needs to scale to a large audience, many servers will be required, and other technologies such as Redis will also be required to set this up properly.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

I’d say the number one design constraint is cross compatibility between different types of operating systems. Other design constraints are things like data persistence, and the scalability requirements not being completely specified.

## [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 below displays a rough overview of the game application. The main class runs everything, and without this nothing will ever happen. The GameService class is like the controller of the application, it manipulates the objects and stores them into data structures to do specific things. The Game, Team, and Player object all inherit the Entity class, but each of these objects don’t actually do anything on their own. They’re like a blueprint for a greater purpose.

**"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** | Requires mac hardcore to run, mac hardware is a lot more expensive. However, macs are very secure and reliable. | Linux is lightweight and is typically used for hosting services. Setting up hosting is as simple as a few installations and getting the service up and running. | Windows is memory heavy, and not light weight. Useful is the software is using Microsoft technologies like ASP or .NET | A mobile device should not host any service if that server is meant to be reliable. |
| **Client Side** | To have mac specific client-side software, the application would have to be re-written in Swift, which is very expensive in time and money. Therefore, a cross platform solution would likely be better. | Linux will not add extra cost to the creation of the application, so long the language used for it matches the language used for the other operating systems. | For windows specific, C# would be great, but it makes more sense to build the software using something that also works on Linux, such as Java. | When it comes to mobile devices, Java is best for Android, and Swift is best for Iphone. Cross platform will save time, but has the trade off of not looking as good as an Iphone or Android specific Application. |
| **Development Tools** | The best language for mac is Swift, with the IDE xcode, although vscode could probably be used as well. | Mostly anything can be used on Linux, Jetbrains, vscode, etc. And mostly any language, just not windows or mac specific languages. | C# is the main language, along with VisualStudio as the main IDE for windows. | Iphone uses Swift, and Android uses Java and more lately Kotlin, which both use Jetbrains. |

## 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**: The hosting should be handled in a Linux environment.
2. **Operating Systems Architectures**: Linux has more control, better security, and is used for hosting services more than any other platform. It’s faster, easier, and overall, the best solution.
3. **Storage Management**: Hard storage should be handled with MySQL or MongoDB.
4. **Memory Management**: Linux is extremely lightweight and can easily handle the memory requirements for Draw it or lose it.
5. **Distributed Systems and Networks**:

The best way to have this application work on multiple platforms is to use a cross platform solution. This heavily lowers the time and cost of development and requires less expertise. Rather than needing a Swift, C#, and Java developer, you could only need one type of developer with a cross platform solution. Java could be an option, or maybe something with better front end options such as Flutter or React. This would give support for a website, desktop application and mobile all at the same time. This is also a lot simpler to run inside of servers, as it’s only 1 application.

1. **Security**: Obviously the best way to avoid security issues is to not store any personal information about the user. If it’s necessary to do this, then using a properly setup Linux environment is the best option.