

Game 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 | 09/21/25 | Adam Fine | New revisions to document different operating systems |

**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 design problem is that there only needs to be one instance of the game service at a given time. This means that there will need to be a unique identity for each game, player, and team. The names of the teams will also have to be unique and need to be checked.

## 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)

Design constraints for this application would include the unique naming, supporting multiple operating systems, having unique naming, and making sure the proper data is shown.

## [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 model shows an entity, game, team, player, and game service. The OOP principles shown are inheritance, encapsulation, and polymorphism.

**"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** | Although Macs offer servers, they are less common due to the cost that it takes to have one. It is mostly for development, rather than hosting. Licensing costs are technically not required because it comes with the physical machine. The cost for a server would be around 50-150 dollars a month for cloud service. | Linux is one of the most used operating systems for servers because it is cheap, has a lot of software it can run, and it very customizable. The downside is that it is harder to troubleshoot. This operating system does not have required operating costs. The costs of hosting though, would be about 350 a year per server. This would be the price if it was ran through Redhat. | Windows is good because it integrates with other Microsoft products, but it is still expensive. It is good to use when companies are already integrated to Microsoft. There are licensing and operating costs for Windows. The OS would cost around 139 dollars MSRP, with the server itself costing 25 dollars per core a month. | These are almost never used for servers because of the lack of specs it has. It is very underpowered compared to the rest on the list. There are not licensing costs because it comes with the physical machine. For apple, the cost to run is 99 dollars, while it is 35 dollars for the Google Play developer application. |
| **Client Side** | Macs are very good for the client because of the UI experience and how easy it is. The only issue is that it is more expensive. | Linux is good because it does not cost anything to run, but it is much harder for the average person to use and does not have all the capabilities of Mac and Windows, depending on the OS. | Windows is the most used because of wide range of compatibility it has. It is still expensive on an enterprise level but is a lot easier to use than Linux. | Mobile is very important because almost everyone has a phone. |
| **Development Tools** | IDEs that Macs can use are IntelliJ, VSC, Eclipse, and X code. All of the software here has a free except for intellij. | Popular IDEs that can be used on linux are VS code, Eclipse, all JetBrain IDEs and more. All of the software here has a free except for some jetbrain IDEs. | Windows can use Visual Studio, VSC, Eclipse, and more. The expenses of the software are the same as Linux. | Mobile devices can’t use much, but they can use Xcode and android Studio. There are free versions of the software the mobile devices use. |

## 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 would personally use Windows because I am more comfortable with it, and I find that it easily integrates with everything. Regarding it being used for a server, I find that it would be easier than Mac and Linux because of the availability it has to most people. The operating system is also very secure when it is properly maintained.
2. **Operating Systems Architectures**: Windows can make applications that support Mac with emulation, other Windows devices, Linux, Android, and IOS, given it is web based. Comparing it to both Linux and Mac, windows being chosen would be for the best.
3. **Storage Management**: Since we would be using Windows, I would recommend using SQL as it works well with Microsoft products. We would pair this with using the storage on the device itself to keep long term data that would be used many times, like saving images, etc.
4. **Memory Management**: RAM would cache the memory and read/write the data in real time. This would be used for keeping short-term information saved but delete once the application or device is powered off. This helps with saving the storage needed within the application for more important data.
5. **Distributed Systems and Networks**: I think that the application being web based would be the best solution as it would be easily accessible, regardless of what machine the client is using. To prevent an outage of a specific component disrupting the entire application, the application itself should be developed into microapps. If one microapp goes down, the others won’t. RESTful APIs should also be developed in terms of the backend as they can be used across all different forms of the User Interface, including phones.
6. **Security**: I think that the application should be made with API keys and Secrets that would be embedded within a hidden cache system that cannot be read by anyone else, other than the server and the client. The application should also use certificates to validate the person using the application is really who it is.