

<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 | <07/13/2023> | David Gerardi | Allow multiple teams, and multiple players per team |

**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)

We are required to make a web-based version of this already existing game. This will require us to use a language like PHP, JavaScript, Python, Java, HTML or many others.

We need to add the ability for one or more teams to play the game

Each team must have the ability to choose a name

We need each team to have one or more players able to join the team

We need the game and team names to be unique when choosing a team name

Only one instance of the game can exist at a time.

## Requirements

The client has asked for several specific requirements to be met:

* Design a web-based version of the game
* 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)

To begin working, we must convert the game room to a web-based program. This may require a new programming language to function properly and efficiently. JavaScript seems like the best choice from what I have seen so far.

Who will be hosting the web environment? Will we be expected to host? Will they be looking for a different host like AWS? Or will they be self-hosting?

What are the hardware requirements for this game? Will it be for mobile devices or PC?

What is the schedule like? How many people should we devote to working on this?

We do not know if the teams being added are required to be even in size, or if we can have an odd number of total teams

## [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)

It appears that for each instance of the game, we’re meant to have a between zero and many teams. And for each team we’re to have zero to many players in each team. Each team will consist of a list of players, and a team id. There should be a function to add the ability to add a player to teams. Each player should have an ID and a name.

There is only one GameService, but there can be zero to many Games that use that GameService.

Game, team, and player all inherit attributes from the Entity class.

The ProgramDriver uses the SingletonTester to test if the program is a singleton

**"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** | The largest advantage for using a Mac is the compatibility with Mac Clients. There should be no compatibility issues and things should be easy to program for if the clientele is also using Mac operating systems. | A large reason to use Linux is because it is open source. This means that anyone can contribute to making it a better project. It also generally means it is free, or significantly cheaper than the other alternatives. There are no licensing fees, and no software/hardware purchases necessary. Linux Kernels are also the most secure and least prone to hackers. The largest downside of Linux is if you are unfamiliar with it, it can be complicated at first. | Windows if the most used Operating System in the world. Using it for a server would ensure compatibility with the greatest number of users. It is very beginner friendly which makes it easy to use. The drivers are always up to date because of the massive team that works on it. It’s guaranteed to have long-term support because of how widespread it is. The issue with it is the high licensing costs, which can increase with each user. It also is the most vulnerable to malware because it is the most used. | Mobile devices require a server that is compatible with many different things. From the massive number of different operating systems used, to accounting for latency. If you’re trying to run a mobile game server, you must account for the fact that some people may have bad reception, or weaker internet signal strength and may receive information slower than the other people playing. Solving that issue is paramount to having a server running a mobile game. |
| **Client Side** | Developing a game for Macs targets a specific audience. While things for other operating systems can be ported to Mac, I believe it is harder to go from Mac to other operating systems. If your target demographic is a mac user, then this shouldn’t be a problem but keeping an open mind about who could use the product is a good idea. | Linux for clients can run with minimal system resources. You can customize it completely to only use what is necessary. It is less prone to malware attacks, like viruses. It is also very privacy friendly because there is no telemetry that is required for you to use it. Other Operating Systems may store your data and use it for things you’re unaware of. The largest downside of Linux is if you are unfamiliar with it, it can be complicated at first. | Most computers in the world use Windows. It is what many people are familiar with. Even if people do not own a Windows computer, there is a very high likelihood that they have used one in the past. If you’d like to hit the largest computer audience possible, windows are the best choice. It’s very beginner friendly. It has the largest range of software compatibility. Because of the massive team that works on windows, drivers are updated very frequently ensuring compatibility. | Trying to account for all the different variations of mobile devices is hard. Things like screen size, operating system, and even screen layout now with folding phones. Trying to make something that looks good for all the different options can be hard. Unlike on PC, window monitors are standardized to 16:9 aspect ratio, phones can be any ratio. They also come with different pixel density so making sure that the text is clear and readable is difficult. While Windows if the most popular PC operating system, I believe there are more mobile devices than PCs. |
| **Development Tools** | There are Apple development tools that are used to build this type of software, but also general IDEs work also. Apple Developer Tools should be used to have the latest support for API references in the apple store. Along with ensuring compatibility with current generation Mac OS. | Linux is able to use a wide array of programming languages from C/C++, Python, Java, JavaScript and much more. Some common IDE’s for Linux are VIM, CodeBlocks, and Eclipse(with the right plugins) | Most programming languages work on windows. C/C++, Python, Java and Javascript, and more! Your choices for IDEs are visualStudio, eclipse, pycharm, Vim and many others | The top 3 programming languages for mobile devices are C++, C#, and JavaScript. The most popular IDEs are React Native, Android Studio, and Xcode. |
|  |  |  |  |  |

## 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 operating system I recommend for The Gaming Room to expand is Linux. It is open source and has the least vulnerabilities to security risks. Since it is open source. There would be no licensing fees, and everything would be customizable to whatever standards must be met.
2. **Operating Systems Architectures**: Linux operating systems use the Linux Kernel. This interacts directly with the hardware and makes sure there is enough memory for available applications, optimizes processer uses, avoids system deadlocks and much more. There are many different Linux operating systems that use the Linux Kernel.
3. **Storage Management**: Since “The Game Room” will be a web-based game, it would make sense to use cloud storage for any information The Game Room would require. This would ensure that there would be no chance of the system going down for any reason.
4. **Memory Management**: Linux can use multi-level page table and huge pages as two memory management techniques. Multilevel page tables are a hierarchical structure to manage memory that has each level representing a different page size. Huge pages are exactly what they sound like, where the operating system uses 2mb page sizes to manage less pages and access memory pages faster than normal.
5. **Distributed Systems and Networks**: We can get The Game Room to function on all platforms by making it a browser-based game. Whether a player is on mobile, windows, mac, or Linux, they all have the capability to open a web browser. Formatting the layout to function on PC and mobile may be difficult but should be doable. This would ensure a frictionless entry into the game by clicking a link, or going to the website and the game would start.
6. **Security**: We can protect a user’s information by ensuring that all the data we keep is encrypted. This requires hash and salt for all passwords and data collected. A hash makes it so that when a user enters information like a password, it is put though a 1-way encryption so that any information we save cannot be reverse engineered to the original password. We also would want to ensure that there is active Anti-Malware software on the server.