

<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 | 1/19/2023 | Jeffrey Carlson | Filled out |
| 1.1 | 2/5/2023 | Jeffrey Carlson | Project 2 |
| 1.2 | 2/18/2023 | Jeffrey Carlson | Project 3 |

**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)

Set up the environment and facilitate the development of the web-based version

## Requirements

Game will include one or more teams

Each team will have multiple players

Game and team names must have unique names

Only one instance of the game can exist in memory, unique identifiers for each game, team, and player

## [Design Constraints](#_2et92p0)

Controls are different on the web versus controls on an android device.

Would have to account for web browsers across all devices capable of using internet.

Use singleton and iteration to make sure that there are no duplicate teams, players, or games

Singleton will check to be sure there are not a duplicate entity and iteration will increment the id of a new entity so there isn’t a duplicate.

## [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 service, game, team, and player can have one or more instances. Program driver calls singleton tester. Entity class is the parent class to game, team and player which are all related

**"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 has a server that costs money to license | Linux is free to use as a server and has a variety of different distros to use for hosting a server | Windows has a server that is easy to set up but costs money for licensing | Mobile devices do not have enough computing power to be used as a server for large clients. Might be good for one person with simple requests sent to server |
| **Client Side** | You need a macbook to develop for MacOS which is pricey. | Takes a while to develop for because of the variety of distros and how each can affect the program to create different bugs | Should use the .net framework for good security and capability. | Developers for mobile need experience because it’s hard to develop for mobile because of the wide range of hardware and how no two phones are similar, each can have different screen resolution and each a different aspect ratio. |
| **Development Tools** | Mac Book with a familiar ide that needs to be coded in swift | I suggest using vscode or visual studio 2022 because both have worked well for me on linux | I suggest using vscode or visual studio 2022 because both have worked well for me on windows | For android applications you can use android studio to develop the application and for iphones you need a mac book that can be used to write swift. |

## 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 recommend using Linux to host the server on because of the variety of distros that can be used. One free recommendation is CentOS because of the length it is supported and is relatively lightweight which lessens the hardware requirements needed.
2. **Operating Systems Architectures**: Linux kernel is stable and secure. One benefit of linux is that malicious actors often try to attack the major OS which is often Windows so that’s another small layer of security.
3. **Storage Management**: I would suggest using ssds because of the speed they offer. IF there is a budget I would suggest running the OS on an ssd and keeping the most used files/programs on the ssd and keeping the bulky storage on hard drives which are slower but often cheaper.
4. **Memory Management**: If the system is being hosted on the cloud you could make it so the server adapts the amount of memory used to match the load at the given time. This might be cheaper than just statically renting the memory needed at peak hours
5. **Distributed Systems and Networks**: All of the user devices would connect to the one server so I don’t think there would be much of a problem there. The big problem would be advantages because of the different controls for each system. If there was an outage with the server it’s possible to have an alternate location for a backup server so that the players can keep playing the game.
6. **Security**: Users information such as passwords should be salted and hashed and stored in a secured database. This would make it hard for people to get access to the database but if they did it would be hard to crack one password and nearly impossible to crack all the passwords.