

<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 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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 goal of this project is to make a web based game with specific functions mentioned by the product owners. Some of these would include having multiple teams, teams having multiple players, and game and team names being unique. Also, only one instance of the game can exist in memory at any given time. These requests make the solution simple, implementing them in the game.

## 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)

There are a few design constraints in the given prompt. First, given were making a web-based design, and there is an Android app version. This means that the web-based design will need to function on different systems, leaving HTTP / HTML as an optimal choice. Next, we need to develop a similar User Interface as the app has, or create a new better one. Then, the next constraint is the expectation that there can be multiple teams with multiple players on each team. Only one instance of a game can exist, so as long as the program and server can handle multiple players, it’s fine. Finally, each team and player need to have a different unique name, this can be checked by checking for an existing ID.

## [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)

a

The UML diagram below gives a visual relationship of the program at hand and how it functions. We can see that the entity class when created has a relationship with the game, the team, the player classes. We can see that these classes inherit from the entity class using similar variables and base functions. Next, in the upper left of the diagram we can see the programDriver and singletonTester classes are related. We can see that the programDriver class uses the singletonTester class to check for the pattern, also allowing us to make sure one instance of the game is active at a time. The next class is the GameService class which is related to the game class. This’ll hold the specific game functions, and how it functions. We can see through the arrows how the different classes interact with one another and hiw they are related. This diagram will help give a clear picture of how to approach the development of this software and game.

**"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** | Compared to Windows Mac runs on less hardware configurations. However, with built in web commands, command line tools, and commands to access the server, these can be seen as clear benefits. | Linux takes much more time to be used effectively, but is a great choice. It has a built in command system, and is less costly compared to its counter parts. Linux is secure, and can run for long periods of time. | Windows offers familiarity to most users, granting easy use and access. It offers SQL support and offers software that can only be reproduced virtually on other Oss. | This specifically requires starting from scratch compared to the other Oss. Holding the server and creating an app that works across all devices takes time. Hosting and creating the backend is not optimal here. |
| **Client Side** | Mac’s ease of use is similar to Windows, fairly easy, maybe slightly more difficult than Windows. A short amount of time would be needed to learn, due to its easy interface. | Linux takes the most time compared to all the Oss as it is seen as the hardest to understand. It is most used by developers because of its open source programs. | Windows is probably the easiest to use of all the Oss. It has many tools built in to support website development, and web app development. | Mobile devices allow ease of access anywhere any time compared to being in front of a computer. Mobile apps should have an easy to understand interface, and should account for differing devices and there Oss(iOS, Android, etc.) |
| **Development Tools** | Chrome development tools, SQL, VSCode, JavaScript, HTML, CSS, Unix, Xcode | Visual Studio Code, Atom, Vim, JavaScript, HTML, CSS, SQL | Visual Studio Code, git Windows, JavaScript, HTML, CSS, SQL, Gvim | In terms of an app, JavaScript and access on both large mobile app store, Google Play and Apple’s app store  In regards to access via website, being able to use different browsers (Edge, Chrome, Firefox, Bing etc.) |

## 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 OS I would recommend using is Windows OS. Windows is the most popular and most used OS of the three. Also, because the game is web-based, Windows is great to use on the backend side.
2. **Operating Systems Architectures**: Windows is the most popular OS in part due to it’s easy user friendly UI. Windows is also able to run different development tools and environments easily (Xamarin, NetBeans, Visual Studio etc.) allowing for optimal development.
3. **Storage Management**: As mentioned before using a cloud server is a great move for the game. This would allow the game to expand as needed and wanted, while also allowing new features to be added and updated. This with windows will allow an easier time on the backend.
4. **Memory Management**: The two options are either virtual or physical memory, with virtual memory being ideal to use. Virtual memory handles larger programs more effectively than physical memory. Also, virtual memory is more secure than physical memory making it even more ideal.
5. **Distributed Systems and Networks**: Given it’s a web-based game there is one definite problem to consider. This problem would be packet loss. The loss of information in communication between a system and network can be frustrating but there are steps to prevent this. Constant communication and work between the development team is essential but also, prevent network congestion through server expansion. This will allow the server to handle more user requests at a quicker pace.
6. **Security**: Many things have already been put in place that will create a secure game. The use of virtual memory is more secure than physical as well as the use of a cloud server that guarantees security. There are other steps that can be taken to promote security as well. Routine check ups on information and security will promote security. Also, Windows has programs in place (and there are third party applications like Norton, bitDefender, and even VPN’s) that help security like Windows Defender, an antivirus program that helps defend from threat actors and malware alike.