

Gaming Service

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_gjdgxs)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_3znysh7)2

[**Executive Summary**](#_2et92p0)3

[**Design Constraints**](#_32hioqz)3

[**System Architecture View**](#_1hmsyys)3

[**Domain Model**](#_41mghml)3

[**Evaluation**](#_vx1227)3

[**Recommendations**](#_1v1yuxt)5

## [Document Revision History](#_3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/19/21 | Tray Bridgewater | Filled out document for the first time |
| 2.0 | 12/01/21 | Tray Bridgewater | Evaluated platforms for client & server side, dev tools |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## [Executive Summary](#_2et92p0)

The client, The Gaming Room, is attempting to port their application from Android to the web. The purpose of this is to enable The Gaming Room to service a larger number of players rather than be restricted solely to the Android market. We will develop a new web-based software environment that allows for games with multiple uniquely-identified players and teams and ensures only one instance of the game exists at any given time.

**Requirements:**

The web app must divide players into one or more teams that each can have multiple players. These teams and players both need unique identifiers that can be checked for availability. The software environment must ensure only one instance of the game is running at any moment.

## [Design Constraints](#_32hioqz)

1. First of all, while no budget was explicitly given, development costs should remain as low as reasonable.
2. There must be a relationship between teams and players wherein the Player class is associated with the Team class. Each team should allow for multiple players.
3. Each player and team should have unique identifiers. There should be ways for the system to easily ensure each identifier is not currently in use by more than one entity.
4. To prevent data leaks and ensure that memory is not being used excessively, we must make sure only one instance of the game is ever running.
5. The software should be well-designed, modular and easily expanded upon. Performance should be maximized and our software should be bug-free.

## [System Architecture View](#_1hmsyys)

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](#_41mghml)

There are two distinct sets of classes in this model. The first one contains the ProgramDriver and SingletonTester. This is where the other classes are actually ran and utilized. The second set of classes are the GameService, Entity and the Entity’s child classes (Game, Team, and Player). Game, Team & Player implement a number of basic attributes and methods from the Entity class. The GameService class regulates and manages the games.

**"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](#_vx1227)

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 OS is a good choice for the server side. macOS Server is ran by Apple and offers a number of important services. That said, macOS Server works best when the clients are also Apple devices. It tends to scale poorly, making it ill-suited for our purposes. | Linux is the best operating platform to host an application server-side because it is tremendously versatile. The open-source nature of Linux has been conductive to an extensive development and hosting ecosystem on the platform. Lacks any central formal support from a major company. | Windows has more options for hosting & maintenance software than Linux or Mac. It is a cheaper hosting environment than Mac, but costs more than the open-source Linux. Microsoft offers some official support like Apple does with Mac. | Mobile devices are exceedingly poorly-suited for server-side hosting. They are not designed to host applications, and most mobile development is done on other operating platforms. |
| **Client Side** | Mac is a fairly common client. While any well-designed HTML browser-based application should work fine, Apple provides an abundance of support to developers and users alike. | Linux is a difficult client to use & its users tend to be power-users. While we don’t need to do anything specific to make sure our program works on Linux, it needs to be well-designed and tested across browsers. | Windows has the advantage of being the most widely-used operating system on the client side. Given that we are developing in HTML for the browser, all major operating systems will be supported. | Mobile is widely used. Our current version of Draw It Or Lose It is designed for mobile, so our primary consideration will be making sure mobile users are able to effectively play with Windows, Mac & Linux users. |
| **Development Tools** | XCode is a frequently-used IDE made by Apple. It is free to use with any Mac product, and is highly regarded by many in industry. | NetBeans is a popular Linux Java IDE. Like most Linux applications, it is open-source. It also allows for HTML/CSS/JS development and has no licensing fee. | This kind of software can be easily implemented using Java on Windows. One IDE commonly used on windows is Visual Studio. The Enterprise Subscription for Visual Studio is $250/month. | One tool for developing mobile apps would be the Oracle Mobile Application Framework, a framework designed for mobile development using Java. The precise cost of an OMAF license varies and an Oracle sales rep would need to be contacted. |

## Recommendations

1. **Operating Platform**: The Gaming Room should consider using Linux as its primary server side operating platform. Linux is open-source and, consequently, has an abundance of well-designed tools for hosting complex services. On the client-side, the application should be accessible by as many platforms as possible. The application should be developed using HTML-5 because of the near-universality of HTML.
2. **Operating Systems Architectures**: Linux is a highly-modular operating platform. There are multiple Linux “kernels” that each have different architectural details, but the broad strokes are the same. The bootloader regulates the OS’s booting process, the kernel handles memory, peripherals and other core systems. The third major component is the graphics system, which handles the output of visuals to the computer’s monitor.
3. **Storage Management**: We should divide our program up into things we want to keep persistent and things we want to be able to change with ease. The persistent parts - the UI, game logic, etc - should be coded into the application itself. Data we expect to change, such as user authentication information and the game images, should be stored separately on an external server.
4. **Memory Management**: Draw It Or Lose It! should use dynamic memory management for reasons mentioned previously, but also because it is the memory management style of choice for Linux. Dynamic memory management is well-suited to our purposes because the number of game instances we expect to have running at any given moment is variable - so we need to be able to allocate the appropriate amount of memory in a dynamic fashion that responds to the program’s particular demands.
5. **Security**: The Gaming Room should consider implementing end-to-end encryption for any sensitive information (such as user passwords). Additionally, the code should be constructed in such a way as to not allow direct memory management or data leaks.