

<THE GAMING ROOM>

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

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

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <11/11/2022> | <Luke Peters> | <Initial Software Development Prototype> |
| 2.0 | <11/25/2022> | <Luke Peters> | <System Platform Evaluation> |
| 3.0 | <12/08/2022> | <Luke Peters> | <Platform Recommendations> |

**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 same developers that are working on this application, they made the game you may be familiar with, Draw It or Lose It. This ideally needs to be a web-based game that has the ability to work different platforms which will work off the current game. The basic concept of the game is to host multiple games who will each allow several different teams which of course will need multiple players. In the end, the game should function by each team having their own name.

## [Design Constraints](#_2et92p0)

Web, IOS, and Android. The unique design constraints are that all these platforms have different software dev kits. There will need to be 3 separate API’s that will need to work with the different platforms. They also should support more than one team from other platforms. The client expressed to our team how important it is to have unique game and team names. Must alert the team lead of a name that is trying to be chosen, that it is taken, they must then choose a different name. Must also use unique ID’s for every player, each team, and then also the game itself.

## [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)

So as we can see from the UML Diagram for The Gaming Room, the main() is ProgramDriver. This is the method that is directly associated with SingletonTester which uses the method of testSingleton that will test and check for the instance of GameService. Next, we can see that we look at the entity class which we can see is the parent class of the game, team, and player classes, they then will inherit the attributes that entity then requires. So, because of this, this makes it possible that Player cannot have a team but then team can have a player. It cannot work that way. Again, it is the same for the team that it cannot have a game but then also the game cannot have a team. It is because a game cannot have the GameService but it can however have a game. Since we are trying to set it up to have unique identifying names, the GameService must have an instance of each game that is functioning at any given time.

**"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** | This OS will function well for servers, but the downfall is that there must be required licensing. | This OS is one of the best for web software. For this situation it is best because it is free to use. | This OS is known for their secure server hosting and ease of use but there is a catch. The licensing can get pricey. | Mobile Phones are not equipped to run servers. While it is known that they can, it is not recommended. Mobile phones are capable of development though. |
| **Client Side** | This OS is a great resource of SDKs but of course to develop with it you must have a Mac. You will need a dedicated developer who has a Mac and knows Swift. | This OS commonly uses Python. This will cost the most in development time. | This OS is the most widely used so familiarity is a big factor here. It has some of the highest requirements. Should use .Net for structure and security. | Mobile phone OS are unique. You need dedicated developers for every different OS. UI/UX should be top priority, this differs from web-based software. |
| **Development Tools** | To write in Swift you at least need to have iCode on a Mac. | This OS comes preloaded with Python. | This OS most commonly uses C++ or C# and can use Visual Studio. | IOS – need a Mac ad needs to know Swift, can use iCode  Android – need Android Studio. |

## 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 believe it would be important to open this up to all platforms in the long run but to start, I am eyeing the security and ease of use of Windows. Unfortunately, you so have to shell out a little bit for licensing and server hosting, but if you can spare the change to develop, I think it would be great in the long run.
2. **Operating Systems Architectures**: This could effectively be developed using C++ while being written in Visual Studio.
3. **Storage Management**: My recommendation for storage would be to use SSD for storage.

1. **Memory Management**: My ideal memory management would be to set up an active watcher to monitor the load of the system.
2. **Distributed Systems and Networks**: The goal of this project is to is for it to be hosted from a cloud-based server so in this case. This works really well when maintenance is needed. The game doesn’t even need to go offline. You can simply switch servers and it will continue to run smoothly.
3. **Security**: From my research I have seen that in this case, role base security could work really well. This gives every role a separation, from the player to the admin, to the game, team, and the individual user.