

Draw It or Lose It

# **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 | 11/11/2024 | Gaidge Jellick | Completed the answers for the assignment |

**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 Gaming Room has an Android app called Draw it or Lose it, a game where teams attempt to guess what is being drawn. They would like for it to be developed in such a way that it could be web-based allowing for multiple platforms to run the game and join in the current sessions. It consists of multiple teams which holds multiple players, where the teams are hosted within a game, which is ran through multiple instances. Because of this, each game and team name needs to be unique.

## Requirements

*The Gaming Room would like for it to be able to be used on the web, android OS, and apple OS.*

*They would like it to run multiple different games at once using different game and team names.*

## [Design Constraints](#_2et92p0)

Android, IOS, and the web all use different development kits.

The API should run on all three platforms.

The API needs to allow any platform to be on teams.

Game and team names must be unique.

Alerts for the team captains that a team name is already in existence, so they must choose another.

Use unique ID’s for each instance of the game, team, and player.

## [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)

The PorgramDriver Class contains the main method, which uses directed association with SingletonTester to test if there is already an instance of GameService. Entity class is the parents class to Game, Team, and Player classes, which all inherit Entity’s required attributes. Meaning that a Player cannot have a Team, but a Team can have a Player. The same goes for a Team and a Game. Where the Team cannot have a Game, but a Game can have a Team. Same thing with a Game and a GameService. Each Team can only have one of each individual Player at one 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** | Macs can be used as a server, but the licensing is expensive and you need Mac books to develop. | Linux is good for web-based hosting. It is the most popular when it comes to hosting and is free when it comes to licensing. | Windows has very secure servers and are easy to set up but the licensing is expensive. | Mobile devices can be used as a server, but are not very powerful making it difficult to host anything off of. They are good for development use though. |
| **Client Side** | Mac is easy to use, but requires a lot of money and someone with the skills to develop in Mac OS. | Most likely the cheapest of options, but requires someone with python skills in order to develop. | Great for making the framework secure, but would cost a pretty penny. | Would need someone who has experience with developing Apps, as a lot of the way things look and are interacted with need to be dealt with differently than on the web. |
| **Development Tools** | Mac typically use iCode. And it is normally done through Swift. | Python is the primary language in Linux, but you could others. Intel also has a great IDE to use for Linux. | Visual Studio Code is the standard when it comes to Windows. This allows for multiple different languages but is primarily C++ and JavaScript. | When it comes to mobile you have multiple options. Androids would require Android Studio to develop apps. Apple would require a Mac Book to develop an app using iCode and Swift. Lastly you could create an app through Unity using C++ and use the Unity converter to have it set to the OS you want, like Android or Apple. |

## 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**: Since the request is to make it web-based I’d probably recommend Linux since it is great for server hosting and web-based development.
2. **Operating Systems Architectures**: You could probably use something like Linux kernel as it is secure and stable.
3. **Storage Management**: SSD would probably be best in this context as it is faster to access assets like images or drawings form the user end.
4. **Memory Management**: Since it is going to be mostly web-based I’d probably recommend a swap partition this way there is no physical memory being used on the user end, and it is all being held virtually on The Gaming Room side.
5. **Distributed Systems and Networks**: Since it is web-based it will most likely be through the cloud, having multiple servers will allow your game to run constantly. So if a server does crash, you can simple have another server activate until the previous server reboots. You can probably do this by routing the servers together, and have the check for crashes, that way if a crash does happen one of the other servers can quickly grab all of the data.
6. **Security**: Role-based security will be best since you want specific people to do specific things. Like the game leader will be the one making the names of the teams and assigning players to each team. By having the Roles, you can make sure that the game leader has this access while the other players do not. Making it easy to also lock players and game leaders out of information that they shouldn’t be able to access.