

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 07/14/2023 | Chance Vosk | Initial Version of documentation |
| 1.1 | 7/29/2023 | Chance Vosk | Evaluation table filled out |
| 1.2 | 8/11/2023 | Chance Vosk | Recommendations filled out |

**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 wants to create a web based app version of their game Draw It or Lose It to serve multiple platforms. Each Game, Team, and Player must be unique within the game instance and must be able to ensure they are unique. The Gaming Room does not know how to set up the environment and has asked us to set it up for them.

## Requirements

* Ability to have one or more teams
* Each team has multiple members
* Game and team names must be unique
* Only one instance of each game can exist at a time

## [Design Constraints](#_2et92p0)

All Games must have a unique name and Id

All Teams must have a unique name and Id

All Player must have a unique name and Id.

Ability to ensure all members are unique and do not already exist in memory.

## [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)

In the UML design below Entity is used as a base class for it’s children Game, Team, and Player. Using Entity as a base class allows us to keep code used across all three children in a place that is easier to update and can still keep code specific to each child. Each GameService can hold 0 to many Games, each Game can hold 0 to many Teams, and each Team can hold 0 to many Players.

**"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 can use the built in Apache web server to host a website for free | Linux is widely used in online industry for web hosting as it is versatile and powerful | Windows does have some server tools built in and can also use open source options, may not be as powerful as linux | While possible to host a server, mobile devices are more suited to only accessing applications rather than hosting |
| **Client Side** | Mac can be very expensive to get into and will require knowledge of how Mac works. Mac can be easier to develop on however. | Linux is a free platform and has a high ease of use once learned. It is generally the least used operating system however | Windows is the most widely used OS and is used in many gaming applications already. It does offer lower security compared to other OS’s due to it's popularity. | Each mobile device could require a different coding language (Apple vs Android). UI designs will have to be adjusted to smaller screen sizes and possibly whole new layout. |
| **Development Tools** | Swift and Xcode are the main things used on Mac and are made specifically for Mac. Translation between other languages may be necessary for Mac to compile correctly. | Can be configured to use basically any language or IDE as OS is open source and many configurations exist | Mainly uses Visual Studio which can be used for a few different languages. Has many tutorials available for its development products. | Each mobile platform will need a different IDE and language specific to it. Could be a bit harder to learn and may need specific training in it. |

## 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**: For an operating system I would recommend Linux as it is widely used for web hosting and can be a very powerful operating system. I recommend this over the others because of its low price and ease of use over the other operating systems.
2. **Operating Systems Architectures**: Linux is an open-source operating system meaning there are many community versions of the operating system. This means that there may be versions of it that are already set up like our needs for the app server. This will speed up development time and as the operating system is open-source it will allow for easier maintenance as code can be easily accessed.
3. **Storage Management**: For storage there is not much storage needed overall for the app including the pictures and user profiles. SSDs can be used for fast access and the price is not too high for small amounts of storage.
4. **Memory Management**: Memory can be managed by passing most processes on to the client side for their memory to manage. Processes like loading the picture can be used on the client side while the server only holds on to items such as the players in the room, the answer for the current picture, and the current score.
5. **Distributed Systems and Networks**: The client can use this server to run across multiple platforms by setting up the server in a way that the server has a translator for multiple different operating systems. Using this any operating system can send requests to the server and the server can send back in the language they received from. This will result in a longer setup time and larger program but will allow for reaching a wider player base.
6. **Security**: User information can be protected in a few ways. The easiest way to do this is through the principle of least privilege. This means that users have access only to what they need and nothing more than that, this can help to prevent users from accessing data they do not need. The easiest way to implement this app would be a role-based system as all players would have the same privilege. Two factor authentication can also be used to prevent false logins of users or admins.