

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 | 08/17/25 | Scott Weiss | Changes made to the Recommendations section. |

**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 client, The Gaming Room, has requested help developing a new web-based game for multiple platforms based on their current game, Draw It or Lose It, which is currently only on Android. Expanding to be used on PC platforms will help expand the userbase for the client.

## Requirements

Business constraints:

* Keep within client budget.
* Stick to required schedule.
* Team assignments, depending on OS familiarity.

Technical constraints:

* Meet standards for web privacy.
* Hardware limitations between Windows and Mac OS, and what each minimum and recommended requirements are.
* Have the game be compatible with latest OS for both Windows and Mac OS.
* Maintaining an up-to-date library of stock images to use.
* Maintain connectivity between users.
* Meet compatibility requirements for multiple web browsers.

## [Design Constraints](#_2et92p0)

In terms of business constraints, keeping within client restrictions is a must, by sticking to the required budget and schedule. Also making sure the team is assigned properly due to experience and needs. For technical constraints, making sure to meet the standards for web privacy is needed. Keeping up with compatibility with both operating systems for Windows and Mac, as well as meeting at least minimum recommended requirements for both. Keeping an up-to-date library of stock images so gameplay stays fresh for concurrent users, as well as maintaining connectivity between separated groups of people. Maintaining compatibility for across multiple web browsers, as users will have many options to choose from. While not necessarily needing every single browser, at least the top five most popular choices.

## [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 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.

Starting with the ProgramDriver main program, it uses the SingletonTester to make sure only one instance of the GameService class will exist in the memory at any time. After this, the program moves on to the GameService class to start a new game, with having the initial private variables of the list of games, GameID’s, PlayerId’s, and TeamId’s. The getInstance() method starts a new GameService by checking whether an existing game is already running and starts a new one if not. Which then gets all of the new variables to start a new game, moving to the Game class.

As seen in the UML diagram, all three classes for Game, Team, and Player inherit from the Entity class. The Entity class will hold the id and name for each subclass for the variables that are created. The Game class creates a new Team list and allows the user to add a new team. With the Game class inheriting from the Entity class, the users will be able to add new teams, to then move onto the Team and Player classes, both allowing the users to create new teams and players respectively.

With using the principle of inheritance, it allows the program to draw from a single class to use the parent class to, in this case, create and hold new variables for new games, teams, and players. Encapsulation is also present in the above diagram, by showing how each attribute is either private (-) or public (+) and how various methods are defined.

## [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** | Hosting options for MacOS is a bit more limited with official MacOS server licenses being discontinued. There still exist third-party options for hosting. MacOS tends to be more secure with an easier interface to use, but the cost of servers tends to be higher than its competitors. | The more popular option for hosting servers. Some plans might only cost a low amount per month, while requiring a smaller amount of storage as well. | While the next more popular option next to Linux, it would require licensing fees for server use, increasing costs, and the cost tends to be higher compared to Linux while also being more resource intensive. | While mobile devices could potentially host a small website, such devices are not built to do server hosting for multiple users. Compared to its PC counterparts, the hardware would be severely limiting. |
| **Client Side** | The time it could take to develop for Mac’s could potentially be longer as the languages such as Swift are not as known. The expertise to code for Mac products is limited, as Mac’s are not as popular as others such as Windows. Given such limited clientele, the cost-benefit ratio of developing software for Mac’s is small. | The languages for Linux such as Java or Python are much more known, so the expertise to develop for Linux would be easier to obtain. However, the benefits would be more limiting than Mac as Linux users are not as common as opposed to other systems. | The languages for Windows being quite common such as Java or .NET make the expertise of developing for Windows easier. Being the most popular OS out of the group, Windows makes sense in terms of business needs to develop for. | For supporting Android devices, a bit more expertise is needed for support and programming. iOS tends to be more streamlined, however needing Mac hardware and software to develop. |
| **Development Tools** | MacOS uses the XCode platform to develop software, while mainly using languages such as Swift or C. For using the developer program, costs range around almost $100/year per developer. | For Linux programming languages, many can be used, such as Java or Python. A Python IDE such as PyCharm is free. For Java, Eclipse is also free. | While not the only IDE for Windows, Microsoft’s Visual Studio is a popular choice for development, which for a professional version costs around $100/user per month. Some of the more common languages supported are Java, Python, C#, and .NET. | For Android, the most common language is Java, while Android Studio environment would be needed to develop the initial program, fortunately being free to use.  For iOS, developing on XCode is needed, which costs about $100/year per developer, using mainly the Swift programming language. |

## 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**: Choosing Linux servers is more ideal for the company moving forward, as it would keep costs much lower compared to the other competitors. Being the more popular option for hosting servers, there are various options to choose from for things such as security software to help protect against attackers.
2. **Operating Systems Architectures**: There will be a backend collection of servers to support and manage the services for the game, while having a frontend to serve as the client access to render the game environment. Putting the rendering on the frontend for the client will help the backend services sustain better processing performance, while also saving a bit on costs.
3. **Storage Management**: Choosing to go with physical servers rather than cloud servers is recommended. While cloud servers may seem like a better option, they can get expensive quickly with longer term costs, and there isn’t as much control over things such as security. While having physical servers might need more on site maintenance and have more upfront costs of acquiring the hardware, there is more control over security and other items that make it the more ideal option. In terms of using SSDs versus HDDs, SSDs in the modern day are the better option, as they have gotten much cheaper for better performance.
4. **Memory Management**: The Linux operating system uses a technique called paging that helps bring efficiency to the system by only loading data when it is needed. In this way, Linux can optimize memory usage and reduce time for starting up.
5. **Distributed Systems and Networks**: The client side and server side would communicate with each other via REST API, allowing the communication to transparent. In terms of connectivity issues between users, users on mobile devices may face bigger concerns depending on network connectivity. To reduce the effect on other players, only allowing users under a certain ping can help limit the overall performance issues.
6. **Security**: From the server side, having only authorized users using SSH keys and strong passwords to be able to access the servers is an essential part, as well as having at the very least a standard firewall in place to act as a layer of security. Having role-based authentication can help protect users by giving all users certain levels of access, mostly being low level access simply needed for game playability functions.