

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 05/24/22 | Earl Calkins | Updated Executive Summary, Design Constraints, and Domain Model |
| 1.0 | 06/12/22 | Earl Calkins | Updated Evaluation |
| 1.0 | 06/19/22 | Earl Calkins | Updated 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 client, The Gaming Room, is looking to develop a web-based version of their game, Draw It or Lose It. This web-based version will work to serve multiple other platforms aside from Android, which the app currently supports. The client is requesting our help to streamline development based on their software and hardware requirements. The specific software requirements requested by the client are:

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

This web-based version will be developed using the java programming language and will need a dedicated web server to host the application. Storage, quick access, and quick rendering for any images will also be necessary. Since the application will be using a web server, security measures will need to be implemented to protect the application. This project will be using libraries provided to us by the client to ensure nothing is left out or unnecessarily added.

## [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)

Starting from the GameService class, instances of the Game class are created to add games to a list of games or get the names and ids of existing game objects. Within the Game class is a list of teams which is populated using objects of the Team class. The Team class also uses a similar method as the Game class but using the Player class to add players to a list. The Game, Team, and Player classes are all subclasses of an Entity class, which initializes the variables used to get the ids and names for each game, team, and player. Since these variables are private within the Entity class, the three subclasses need to use getter functions to access the values assigned to those variables. These four classes (Entity, Game, Team, Player) all also include a toString function, which is used to override appropriately depending on which object class is being outputted. All of these objects are used in the ProgramDriver, where the main() function is. Objects with these different classes are created an outputted, with similarly named objects (such as game names) being set to equal values. The singleton tester is then used to ensure there is only a single instance of each object of the GameService class.

**"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** | Much like Windows, Mac only offers one option for OS. Mac-based servers have low-level base security, which can be increased through manual customization. On the customization end of things, Mac is lacking may be lacking but makes up for it in usability. Mac-based servers may be the most stable out of any OS, as they’re intended to be used on Mac machines. Similar to Windows, Mac-based servers are quite costly. | Linux offers a wide variety of distributions for hosting web-based applications. Linux-based servers are also the most secure of any OS without extra protection. While Linux may be more complicated, it is the most customizable OS. Along with customizability, Linux-based servers are known to be very stable and rarely have stability issues. When it comes to cost, most Linux distributions are free while others are on the cheaper end. | Windows only offers one OS for hosting servers on. Base level security is lesser than Linux-based servers, but security can be increased with extra security management. Windows is a simpler and more usable OS but is much less customizable. Additionally, Windows-based servers have been known to have stability issues. The costs for setting up a Windows-based server can also be a bit pricier compared to Linux-based servers. | The OS used to host a server from a mobile device will depend on the device being used, so there are several options. Security for these servers will be based on the security setup for the mobile device being used. As some mobile devices are more customizable than others, choosing a device that suits your needs is important. Server stability will be based on the power of the mobile device and its signal. The cost for servers set up via mobile device can be free or cheap if using an app from the app store. |
| **Client Side** | When it comes to multi-client support, Mac offers little. Mac-based servers are often made to be used specifically on other Mac products. Due to this, not much can be done to make Mac-based servers communicate with clients using a different platform other than Linux. | Since Linux is so versatile, clients using any platform can communicate with Linux-based servers. It would be quick, simple, and cheap to implement communication with other platforms. | Similar to Linux-based servers, Windows-based servers can also communicate with clients using different platforms. Due to the nature of the Windows OS though, it may prove to be more costly to implement cross-platform communication. | Depending on the mobile device used, communication with clients using other platforms may vary. A mobile device using Android’s OS may be easier to communicate with other platforms than a mobile device using iOS. This would require knowledge on the most versatile mobile OS to choose the best one. |
| **Development Tools** | Developing a web application for Mac requires one main set of tools made specifically for Mac applications. The Swift programming language would be used and XCode would be used as the IDE. Both of these tools are free so there is no cost in using them. Splitting into separate teams may not be necessary for Mac applications, as both client and server sides would likely be written in the same language. | Developing a web application on Linux would use similar tools that Windows would use. HTML using Visual Studio Code and JavaScript using Eclipse IDE and a relevant framework like Angular. There are no costs for using these tools unless a premium version is required. Just like Windows, teams could be made up to support the server side of the code and the client side of the code. | There are several options for tools and languages for building this type of software on Windows. Perhaps the most relevant languages would be HTML using Visual Studio Code or JavaScript using the Eclipse IDE and a framework like Angular. Since these resources are free, there would be no cost to using these tools. Different development teams may be needed to program the client and server sides of the application. | Mobile devices would not be a suitable environment for building a web application. |

## 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 would recommend that the client use the Linux operating platform.
2. **Operating Systems Architectures**: Due to cost, flexibility, security, and stability I believe a Linux-based server may be the best option. A Linux-based server would be simpler to use when attempting cross platform client communication as well. The client would be able to choose a Linux distribution that best suits their needs and the needs of the server. Due to the open-source nature of Linux, the client could customize their experience as needed and use tools and languages familiar to them at little to no cost.
3. **Storage Management**: The best system to store image and user data would be cloud storage. User data could be uploaded to the cloud from the program and back when needed, and also be stored on the client’s physical drive. Image data could be transferred from the physical drive to the cloud for the server to access when needed. This method would have both cloud storage and the physical drive serve as a sort of backup if there are issues with either.
4. **Memory Management**: In Linux, physical memory is separated into three different zones (ZONE\_DMA, ZONE\_NORMAL, ZONE\_HIGHEM) depending on the amount of physical memory used. Draw It or Lose It will likely use the ZONE\_NORMAL region since it should use a relatively average amount of memory, which is anywhere between 16 and 896 MB for this zone. This memory can then be allocated either statically or dynamically by default, or a specialized allocation method can be used. As far as virtual memory management, Linux manages this by creating a series of pages of virtual memory that can be loaded and swapped in and out from the disk.
5. **Distributed Systems and Networks**: Out of each of the operating systems, Linux is the most versatile when it comes to communication between different platforms. Since Linux is a UNIX-based OS, it can easily communicate with Mac clients, and it can implement certain protocols to communicate with non-UNIX-based OSs like Windows. From there, connectivity will depend on factors acting on the server or the clients. Connectivity could still be interrupted by internet outages or lack of network connection.
6. **Security**: The client could implement Dropwizard using REST-style programming to protect user information across multiple platforms. Using this, the client could require that a user inputs a username and password to access their own information but to keep them from accessing others’. This, along with Linux’s authentication and access control methods will ensure that user data is protected from potential threats.