

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 03/09/23 | Patryk Perkowski | Initial executive summary and design constraint of draw it or lose it. |
| 1.1 | 04/01 | Patryk Perkowski | Adding summaries of Mac, Windows, and nix evaluations, exploring pros and cons of each. |
| 1.2 | 04/15 | Patryk Perkowski | Concluding final recommendations for app. |

**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)

Draw it or Lose it is a game designed after a game show in the 1980s where a competitor would draw a picture with an easel, as they drew the other players would try to guess what the picture was. Our implementation of the game currently runs on android, and we are building the web app version that must have the same functionality as the app version.

## Requirements

The game should contain the following: Ability to have multiple teams, multiple players per team, unique game and team names, and only one instance of the game can exist in memory.

## [Design Constraints](#_2et92p0)

The constraint will be to use singleton designs to ensure only one instance of the game exists in memory. Also, development should ensure team and player names are unique, by using singleton design patterns.

## [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 ProgramDriver classes uses a singleton design pattern implementation on the SingletonTester class, which drives the program allowing for only one instance to exist in memory. The GameService, Game, Team, and Player classes all have a 0 to many relationship with each other, and they all inherit from the Entity class. This implementation allows for multiple instances of those classes to be used in that relationship.

**"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** | Yes, Mac’s can use a variety of web servers but the most common are Apache and Nginx. However due to the closed source environment things such as patching, and support could be more difficult. Also, there is a cost associated with licensing, depending on the number of users and devices. | Yes, Linux is the ideal webserver hosting OS for most applications because its flexibility, reliability, and security. Apache webserver and Nginx are both great options on Linux. Being open source there is no cost associated with running a webserver, and there is a vast community of developers that contribute to its development. | Yes, IIS Web server is used on windows. Windows server comes with licensing costs based on number of users and devices. Its benefits include, easy to use and learn, support for .NET, and integration with other Microsoft products. Often enterprises will use this for things such as SharePoint and Exchange. | No, mobile devices are not a practical choice for hosing a webserver. Although it is possible, their a few major cons that eliminate it. Limited resources will not allow any scalability, or just not have enough resources to start your app. Also, mobile devices have less security mechanism, so can easily be targets of threat actors. Lastly, no ethernet options realistically means low bandwidth, and mobile data will not be able to support traditional server needs. |
| **Client Side** | Development cost can be higher due to the need for specialized hardware such as macbooks, and Xcode licenses. The time and expertise will likely be the same since OS client-side web apps are running similar cross platform languages. | Development cost will be the lowest since there are many open-source frameworks available. Linux can easily be virtualized as well. Development time and expertise are like other platforms because of the usage of cross-platform languages. | Development cost will be like Mac since licenses for OS, and dev tools are not free. Development time and expertise will be similar to the others, especially because windows has a strong dev community. | Development cost can be like linux since theirs a variety of open-source frameworks that support a lot of devices. Time and expertise might be a little higher since developers will have to put more effort into testing their web app for responsiveness, screen size, etc on a lot more devices. |
| **Development Tools** | The languages used will be the same for all four, they are HTML, CSS, and JS. The most common development tools are xcode, vscode, and homebrew package manager.  Licensing costs for xcode are 99$ for the developer program but has a free option. VScode has an enterprise license that includes azure devops credits, and other things, but the core product is free. | The same languages as Mac. Some of the most common dev tools are vim, vscode, and git. Vim is free and open source; you can make custom forks of vim for your project that include and special plugins that make your developer workflow faster. | Same languages as the last two. Dev tools for windows are Visual studio 2022, vscode, and git as well. Visual studio comes free but offers a business and enterprise plan being 45$ and 250$. | Languages for mobile device web app development include frameworks such as react native, which uses js and react. Some dev tools include vscode, atom, and Xamarin. Atom had a premium license cost of 139$ but is slowly getting discontinued. Vscode will be the ideal choice for 90% of developers. |

## 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**: Based on the requirements of Draw It or Lose it, I recommend using Linux OS to host the server. Linux is the most flexible, and easy to use option. Being open source, it works well with all different types of software and hardware. Linux provides easy implementation of horizontal scaling to improve app performance during peak hours.
2. **Operating Systems Architectures**: Linux uses a monolithic architecture; this allows for all the core components of the OS to be managed by the kernel. This allows for great flexibility because operating systems can be built around this to fit a specific use case. However, Centos or Rhel would be a pretty good choice, they come shipped with all the required packages for a web server, as well as with a variety of security benefits.
3. **Storage Management**: Storage management has a lot of options, firstly we would have to consider any compliance regulations. Things like length of time cold data must be stored, and whether the cloud could be used, or is a on-prem solution required. If there are no compliance limitations, then I would recommend Amazon S3. AWS provides 99.9( 11 9’s)% availability, and some of the best pricing/performance metrics on storage. It’s also easy to integrate with since there is an aws sdk for all the most common languages.
4. **Memory Management**: Linux’s lightweight kernel allows you to get the best bang for your buck when it comes to available memory. It also includes some features like virtual memory to prevent application for bottlenecking during peak hours. Linux memory management solutions are also designed to be more scalable, allowing greater support during peak hours.
5. **Distributed Systems and Networks**: Using a cloud provider like aws can make distributed systems and networks easy. First, the main server running the application can run on an auto scaling group in ec2 or deployed in a Kubernetes cluster in eks. Load balancers can be added to the mix to ensure servers have equal amounts of traffic going to each one. Cloudfront can help stack live on the edge, allowing users to have super lower latency wherever they are in the world.
6. **Security**: When the app goes into production on the server, it should run under its own user account that has the least number of privileges it needs to run successfully. The operating system should have SElinux configured with appropriate modules, and security policy. At the least /var/\*.log, and application logs should be sent to a siem internally or third-party MSP. Lastly the server should have set up proper encryption certificates from trusted CA’s.