

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 | 11/17/2024 | Mac Grier | Initial summary and constraints added |
| 1.1 | 12/1/2024 | Mac Grier | Added evaluation table |
| 1.2 | 12/15/2024 | Mac Grier | Added recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is looking to create a web version of their game “Draw It or Lose It” based on the 1980s television game Win, Lose or Draw. This game uses any number of uniquely named teams and uniquely named players on each team. By expanding their game’s audience to anyone with a web browser, The Gaming Room is looking to increase their customer base considerably.

## Requirements

Technical Requirements:

* The website allows one or more teams
* Team names will only be accepted if they are unique
* Each team must accept any number of players
* Player names in each team must be unique
* Only one instance of the game can exist at a time

Business requirements:

* The application must work in web browsers
* The most commonly used browsers must be supported

## [Design Constraints](#_2et92p0)

* The game must work seamlessly across each browser
* The user interface should be reminiscent of the mobile version
* Clues must come from the same database of images the Android version uses
* Network delay should not cause issues with the game
* Accessibility features like screen reading should be supported
* All controls should function on touch screen laptops and tablets

These constraints should help developers create a well-rounded application that meets the expectations of a modern user. If The Gaming Room decides to add a timeline or budget, those would also be important things to consider. Developers can also get an understanding of what kind of technical decisions they should make when creating the application and can understand how they can meet the standard of quality that is desired.

## [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 UML Diagram shown below uses seven classes to manage a game. The classes Game, Team and Players all inherit from the Entity class. It is shown that the program has many-to-zero shared objects between the GameService, Game, Team and Player classes. In this program, this is exemplified with the ability to add players to teams and teams to a game. These all have their own IDs and names which can be used to find them later. The GameService class contains a list of all games and utilizes the singleton pattern design so that there is only one instance of it running at a time. The ProgramDriver class contains the main method and is used to test the game. Alongside it, there is SingletonTester that tests to make sure there is only a single occurrence of GameService.

**"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)

Here I have created a table featuring the pros and cons each platform provides in terms of server and client development, as well as what tools would be needed to work with that platform. Prices listed below are subject to change over time and should only be taken as a guideline.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | After the removal of support for macOS Server, any machine running macOS Monterey can host a server. This can make it easy for the client to host a server themselves by buying or using their existing hardware, but it is difficult to find a host service without spending considerably more than for the alternatives. | As the most popular operating system for hosting web servers, Linux offers minimal cost and maintenance with their open-source platform. Linux servers require more experience with the less popular operating system and technical support will be diminished compared to alternatives. There are also potential hardware conflicts if the server was to be hosted by the client. | Hosting on a windows server is extremely beginner friendly with a graphical user interface as opposed to the Command Line Interface Linux uses. If there is a need to integrate with any third-party or Microsoft applications, they are almost certain to be available for Windows. Licensing fees are higher, and the performance of the operating system is diminished by having to render a GUI. Their customization compared to Linux also suffers due to settings that are set at Microsoft’s discretion. | Mobile devices are generally not suited for server hosting. It is possible but the costs and time to create such a system would likely not be worth the trouble. The positives for it would be that the servers themselves could be owned and operated by the client easily. Compared to typical server architecture, it would be extremely limiting for future expansion and performance would be an issue with any expansion. |
| **Client Side** | To develop for Mac, it would require a computer running an up-to-date version of the operating system. The cost would be moderate as Macs are more expensive and prebuilt compared to building a Linux or Windows machine. Languages usable on Mac are less likely to be known by most programmers and will incur a higher cost for development in terms of time learning or hiring expert knowledge. | To develop for Linux, any personal computer can be used. It supports the most common languages like Java and Python which means that development can be done by less expert developers and the time it takes should also be lessened. | To develop for Windows, any personal computer can be used but there is a cost to a license if needed. Development can be done with most languages so lack of language comprehension should be a minimal factor in development time. In addition, Windows is the vast majority of the market share which means that expert talent should not be needed. | For Android, code would be developed in Linux or Windows. There are many free development tools but they are less commonly taught for mobile application creation so there would need to be time spent learning or money spent hiring expertise.  For IOS, it would require Mac hardware to be able to develop for. This comes at a decent expense and requires experience in developing using the Mac operating system. |
| **Development Tools** | The Mac SDK comes as a part of the Xcode IDE bundle and mainly supports their proprietary language Swift and Objective-C. Xcode costs 99 dollars per year to be able to publish but until that is needed, it is free. To work with a team of developers, something like Xcode cloud can be used to organize and synchronize the project. | Due to its open-source nature, there are a multitude of different free IDEs and tools available for Linux. Eclipse is free and does all the languages likely to be needed in a project. These include C, C++, Java, Python, and more. | Windows is a very common operating system to develop in given its share of the market. Microsoft’s languages .NET and C# are likely to be used if this platform is chosen. Their IDE, Visual Studio, is very powerful and allows for free plugins and other languages if needed. Visual Studio will cost up to 300 dollars per user depending on the features needed. | For Android, development can be done in free IDEs like Android Studio or something like Visual Studio if there is a need for a robust IDE. The most likely language to program in would be Java as that is the standard for Android.  For IOS, Xcode on an Apple device can be used to code the application but will require a fee to publish, the same as for Mac. Languages used would be SWIFT and Objective-C. |

## Recommendations

1. **Operating Platform**: I would recommend a Linux-based server system. They are widely supported by hosts worldwide and are cheaper to set up compared to a comparable Windows server. Performance is very unlikely to hinder the application as Linux has very low system requirements but can still scale into a larger system if that is required in the future.

Linux has no problems integrating with other operating systems which makes it great for our environment with the vast majority of users utilizing Windows or macOS. Linux has proven itself by powering some of the most popular and impressive server-client applications like ones created by Google and Amazon.

1. **Operating Systems Architectures**: For Draw It or Lose It, I would recommend a server-client operating architecture. It is perfect for ensuring that all clients are working with the same information and minimal hardware requirements are put on the user. It also easily allows for an expansion of the number of players at any one time by handling everyone on a central server.

In addition, the security measures that are likely to be utilized like role-based access or user authentication work best on a server. There is less risk of data being lost to any individual client being compromised, which is a huge benefit.

Since the application is also being managed on the server, the users will be able to use a modular lightweight user interface. They can connect from any operating system or browser if it follows the standards we set.

1. **Storage Management**: Cloud storage if the server is hosted by a company like Amazon or Azure will be an easy choice as it is scalable, cost effective, and unlimited. The negatives of potentially slower access are not a worry as Draw It or Lose It is not a program that requires split second response time from the server.

If the game is hosted on a local machine, DAS or direct-attached storage will be easy to connect using a SATA cable to hook up a hard drive. This is much more cost effective if there is only a small volume of needed servers. If there the number of servers operating locally grows significantly, on-site cloud storage could also be an option using something like OpenStack.

1. **Memory Management**: Linux memory management is done using caches of data that improve performance significantly. Whenever data is read it is stored in the cache, and it can be easily accessed again using this cache. This cache is dynamic and opens up space whenever memory is needed which ensures that the system stays fast while having flexibility to process. More of the ram is being utilized at any one time which maximizes the usage of the system the server is running on.

The need for better RAM on this project will be minimal however, as there is minimal processing needed server side for the displaying of pictures for the user. For the users themselves, there is also not much RAM needed as only a few images will be in memory at any point.

1. **Distributed Systems and Networks**: Using cloud architecture, the stability of the network is almost guaranteed. All of the data will be backed up, and if there is an issue where we are hosting the servers, it is easy to replicate the application onto servers in a different location leading to minimal downtime.

To connect to the server, RESTful API will be used with standard HTTP methods. This architecture will be perfect for the lightweight and scalable system we are looking to create. It is a breeze to work with and features uniform naming structures and uniformity which will ensure high productivity from the team creating and maintaining it.

1. **Security**: Role-based access control or RBAC assigns permissions to roles and then users will be assigned one of those roles which will give them access to commands and data unique to that role. This improves security for the client by ensuring that users without administrative permission will be unable to access critical systems and vulnerable information.

The scalability of role-based authorization is also a huge advantage with the ability to add more roles and change a role’s privileges easily. In many countries there are regulations that require strict access control systems. Role-based authorization easily meets these while still be simple and lightweight to implement.

Code obfuscation will be done to make it harder for malicious players to reverse engineer the application and expose weak points. In this same vein, user input will also be sanitized to ensure that any “injection” style attack will be squelched.

All of these things, in addition to regular monitoring and auditing of the application and server will prove invaluable to the security of client and the user.