

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 | 09/21/2025 | Anthony Ashcroft | Initial creation of Software Design Document |
| 1.1 | 10/05/2025 | Anthony Ashcroft | Completed Evaluation Table of Document |
| 1.2 | 10/18/2025 | Anthony Ashcroft | Completed Recommendations of Document |

**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)

Our client, ‘The Gaming Room’ has requested assistance in developing a web-based version of their game ‘Draw It or Lose It’ that will support multiple platforms. This game already exists as an Android application but supports no other platforms.

## Requirements

1. Web based product
2. Unlimited games running across multiple platforms
3. Uniquely identified games, teams, and players
4. Game drop in and out
5. Safety

## [Design Constraints](#_2et92p0)

1. The product must be web based: The final product should be able to run across multiple browsers and operating systems.
2. No limitations on number of games running; As the product will be running on multiple browsers and operating systems, it will be required to allow for many instances of the game to be run to accommodate all users.
3. Games should all have unique identifiers: As there will be many games running, it is important to ensure that each game is given a unique (seed) that identifies that game to users. Unique identifiers should also be provided to each team and player.
4. Allow for drop in and out: Users should be able to join or leave the game at will. Using the (seed) mentioned above, it should be possible for users to be provided with the seed to jump into an already existing game.
5. Safety: It is important to implement a method of monitoring user input to ensure that the environment is safe for all who participate. Examples would include profanity filters, and user reporting.

## [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 GameService class will be used to as the server for all games that will be launched for ‘Draw It or Lose It’. It is responsible for managing each unique game along with the teams and players involved in each game. It is designed using the Singleton Method so that there is only ever one instance of the GameService at a time. The current games running is maintained in a list so that they can be searched through by the methods defined in the class. The list cannot be directly accessed and is a form of abstraction as it can only be accessed using the public methods provided. This prevents tampering with the important information that it houses. There can be only one GameService but 0 to many games.

The Game class contains a list of teams that are participating in the current game. It allows for additional teams to be added. The Game, when created, will be instantiated with a unique Game name and Game id. For each Game there can be 0 to many teams. The Team Class contains a list of players participating in the team. The Team, when created, will be instantiated with a unique Team name and Team id. For each team there can be 0 to many players. The Player class is created for each player participating in the game. They have a unique id and name. Each of these classes being separated represents encapsulation. Though they could all be combined, it would make for a difficult to read program, and small changes may have unintended effects across the entire application rather than just the class.

The Entity acts as a superclass, and the Game, Team and Player extend it. This is an example of inheritance. It contains the name and id which is utilized in the three subclasses. This means that the subclasses do not need to have their own declarations of these variables. Because these variables are housed in the superclass, creating subclasses that use this data is an example of polymorphism. Games, Teams, and Players are all entities but are unique classes themselves.

The ProgramDriver class houses the main() function. This will be responsible for launching the GameService and all other classes used in the game. It uses the SingletonTester Class to ensure that only one instance of the GameService Class is reported to exist at a time.

**"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 is built off Unix. Running a server is user friendly but can become expensive. There are not dedicated servers for Mac but the server app can be purchased for $19.99 that adds this functionality. Depending on needs, server ranges are the same as the product itself from a mac mini at $599.00 to a mac studio starting at $1999.00 and increasing. | Linux is an open source operating system that can be run on most machines. It allows for scalable servers and flexible operating system developments. There is a lower overhead for running a Linux server which would result in more bang for the buck. Linux cloud servers start from $44.00 month that increases to scale with needs. | Windows is the most used operating system for server, web, and application hosting. The windows server license scales based on number of cores to use. This pricing can reach up $4,200.00. There are also cloud based solutions that start at $200. And scale to the clients needs. | Mobile devices are generally not equipped to run servers. Most servers for mobile systems are hosted on Windows and Linux distributions. Those costs have been provided in the previous portions of the table. |
| **Client Side** | Due to the nature of mac machines, their product is expensive. There would be a lower user base if the product is not developed with a web based client in mind.  As this is an HTML interface, this should not cause any additional difficulties. | Because the operating system is open-source, anyone can use it to play products developed for it. Due to the complexities of the OS and the many different distributions, the user base is much smaller than macOS and Windows.  As this is an HTML interface, this should not cause any additional difficulties. | As Windows is the most popular Operating System, there are always cutting edge version of web based utilities to facilitate users through an HTML interface.  As this is an HTML interface, this should not cause any additional difficulties. | As mobile device environments are far different than their PC counterparts, it is important to develop a client that is OS agnostic. It would be important to make sure developed clients for mobile platforms use the HTML interface not only to talk between other mobile devices, but also to PCs that have a completely different platform and work environment. |
| **Development Tools** | Apple’s native development environment is Xcode. This allows for the building of tools in Swift, Objective-C, C, and C++. Other languages are available that will run on Apple such as Java | Linux software is generally built for C and C++. Other development platforms are available. As Linux is an open source operating system, there are no specialized development tools. Eclipse is one option for Development | Windows provides their own tool in Visual studio. It supports a large scale of programing languages including, .Net, Java, Node.js, Python and more. | Android has its own development studio called Android studio. This platform uses Kotlin as its primary language.  iOS uses Xcode as its primary IDE. Coding supported in its editor is Objective-C and Swift |

## 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**:

It is our recommendation to use a Linux OS as the server to support facilitation of access and data provision for the clients. It is a low cost, highly scalable solution that is platform agnostic. With this solution you will have the option of hosting your own servers or offloading the work to a cloud-based solutions provider like Amazon Web Services (AWS).

Linux is a strong candidate due to not only its security but also its flexibility. Due to its lower overhead, costs of running a server in this environment are reduced. Linux servers are also reliable, employing the option of an immutable distribution. This means that the core of the server will not change resulting in less downtime to support client needs.

This platform will support API, HTTP, and other web-based server commands to interact with client solutions. It will also be able to support the client platforms you wish to develop for.

1. **Operating Systems Architectures**:

The base of the Linux system is the Kernel. The Kernel handles Memory, Process, and Device management as well as Security, Resource Allocation, and the Security of the operating system. Selecting the right Kernel for the needs of the ‘Draw It or Lose It’ software will be important in its success. For use as a Server, we will be using a monolithic kernel to support your needs.

The next level of the Linux system is the Shell. This is the layer in which the developer or system manager will communicate with the Server to run commands to maintain the software or operating system itself. It is recommended to use BASH as the shell as it is the most widely used interface.

The final level is the application itself. Ensuring that the developed application runs on the selected Linux Distribution is important to the success of serving multiple clients. Though the Kernel handles most of the low-level work, we need to ensure that the application is optimized to utilize the resources available to it.

1. **Storage Management**:

It is recommended that a flexible large scale storage system be used to maintain the assets of the application ‘Draw It or Lose It’. Having multiple HDDs in an appropriate raid configuration can provide not only quick delivery of application assets, but redundancy to reduce system downtime due to error. Users, teams, and games can have their details stored in a quickly queried database stored on a separate drive space not accessible to the client.

Hosting application assets on the server will allow the client to have a smaller footprint and have the assets it needs streamed in. Alternatively, the server load can be reduced by holding only what is needed to run the server and provide the commands needed to the client. Either option being viable, it is important to consider client needs. It is recommended that the server provides data and assets on a as needed basis. This will allow more clients to be capable of using the application due to reduced storage on their end. This will increase costs on the server side due to increased storage needs, but will have a better return on investment as the application grows in popularity and use.

1. **Memory Management**:

As the server will be hosting many clients, a shared memory approach would be effective in providing assets to the clients. For games currently running, assets could be stored in the Server’s memory that is shared to all clients in the running game. This reduces the memory footprint of each game running. Any assets not being used can be offloaded to virtual memory for quick access as needed to be streamed into the main RAM of the Server. By using compression techniques, we can reduce RAM used by application and data delivered to the client. This will place the load of decompression on the client allowing for the Server to support more clients before needing to scale up.

As the application should always be running, enough ram should be allocated for it to always be running to offer games to clients. As clients increase, it will be important to scale the use of server resources to support them. As the list of players, teams, and games are not time sensitive, they can be stored on the HDD for access as needed reducing the footprint of the RAM. A paging system can be implemented to increase the speed of access to the slower mechanical storage of the HDD.

**Distributed Systems and Networks**:

To allow for the most consistent uptime of the application ‘Draw It or Lose It’ we recommend the use of a cloud-based hosting solution. They can shift the server to another location in the event of issues with the current hosted location. As the needs of the system increase, the cloud solution can allocate these resources without bringing down the servers for extended periods. Using a clustered approach will optimize uptime and client satisfaction.

As we will be supporting multiple client platforms, using API or HTTP commands to communicate with the server will allow for a platform agnostic client/server relationship. Having the Server provide what is needed to each client based on the API or HTTP payload means that the client will only receive what it needs when it needs it.

1. **Security**:

The Linux OS Server environment includes firewalls, access control, and is built with a least privilege model. This means that access to the Linux system is granted on an as needed basis. The environment also uses a Kernel hardening practice which means that even under attack, there are limited things that the attacker can do to the compromised system. Access to the Linux server can be limited to SSH to ensure that only the proper level of user can access the server directly.

Access to the Server via the client will be limited to encrypted HTTP or API protocols. This includes Trusted Certificates and SSL/TLS 1.3 encryption. Further encryption will be provided using SHA256 to ensure that user data matches what is in the database. This will ensure that not only client data will be protected, but the sever will also be protected from data thieves. Further, it is possible to provide a form of authentication token such as OAuth 2.0 to allow users to access the server using their client without needing to constantly log in. It is also possible to set an expiration on these tokens requiring the clients to log in regularly or if they access the game from another machine. Tokens also help to reduce server storage requirements as a bonus.

With the recommendation of having a cloud service host the server, there is an additional layer of security. Companies who host cloud services employ their own methods of security with additional firewalls, encryption, and VPN tunnels to prevent attacks from outsiders. They will also ensure that the servers will always maintain the most recent instances of security to ensure that you will be confident in using their services.