

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 09/22/2024 | Alex Hitchens | Original Document |
| 2.0 | 10/06/2024 | Alex Hitchens | Added Evaluation of Development Requirements |
| 3.0 | 10/20/2024 | Alex Hitchens | Added 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 Gaming Room is producing a game titled “Draw it or Lose it”. The game will have players on one or many teams trying to guess a phrase, title, or thing by pulling stock images to use for clues from a library. Each game will be four rounds lasting a minute each. The playing team has 30 seconds to guess, then the opposing team has 15 seconds to steal if the first team fails to guess. The application will be web based. The main challenge for the client is the creation of software to manage these games and facilitate players’ experience while playing. The software will also need to identify previously created names so there are no multiple game rooms.

## Requirements

1. A game will be able to facilitate one or more teams playing.
2. Each team will have multiple players assigned to it.
3. Game and Team Names must be unique and allow players to check whether a name is in use when choosing a team name.
4. Only one instance of the game can exist in memory at a time.

## [Design Constraints](#_2et92p0)

1. The solution must include server-side software to track current games as well as their players and status. This server-side software should also be able to identify unique games and if a player attempts to name a game the same things twice, prevent it.
2. The solution must have security to attempt prevention or report cheating or harassment by players. There is no communication listed, so the only venues of harassment would be through image choice and naming of players and game rooms. Having automatic processes in place to stop inappropriate names as well as a report system will help prevent this. Cheating through means outside the game would be harder to account for.
3. The solution must have a viewable lobby for players to create games or join games. Players need some way to identify games their friends may be playing and join appropriate team/games.
4. The solution must facilitate the game rules laid out above and have screens to display results to the player. The software needs to be able to run the four rounds of each game, pull images from a stock art library and facilitate guesses for both teams.

## [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 shows the overall structure of the system. Entity is a non-referenced class that serves only to provide common elements to Game, Team, and Player through inheritance. The UML diagram also shows the relationship between the four other elements. There are “zero to many” relationships between each of the classes going in order of GameService, Game, Team, and Player respectively. Also, each of these elements has a list of the lower element, Meaning Team has a list of Players, Game has a list of Teams, and GameService has a list of Games. All individual variables are marked as private, the only way to access them are through the various getters and setters that each class contains. Also contained with in this diagram are the ProgramDriver and SingletonTester that serve as a driver for the application as well as a class to test the singleton design type of the program.

**"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 must 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** | iOS is not a common server hosting platform, but it can be done.  **Pros:**  -Good development environment for making software for iOS devices.  - Support for Unix based commands and tools  **Cons:**  -High Hardware cost  -Not designed for large scale server uses.  -Limited support compared to Windows or Linux | Linux is known for flexibility, support and performance for server-side applications. Server distributions like Ubuntu Server make the process easier.  **Pros:**  -Open Source, reduced licensing costs.  -Strong Security  -Stable  -Scalable  **Cons:**  -May require technical expertise.  -Not user friendly as alternatives (Windows or Mac) | Windows Server can be used to web hosting applications. Has a GUI which makes interactions easier.  **Pros:**  -Integrates in windows services like .NET  -Familiar for those already using windows.  -Decent Support  **Cons:**  -Has associated high Licensing costs  -Less flexible than Linux | Mobile devices are not and should not be used for hosting web applications.  **Pros:**  -Mobile platform can be taken anywhere  **Cons:**  -Not suitable for hosting server applications.  -Lacks power and infrastructure  -Would require proprietary software to properly work for The Game Room |
| **Client Side** | **Development Considerations:**  Safari is the major browser on iOS to worry about compatibility with Chrome and Firefox much less so.  **Cost:** Web based client development has no costs but if app needs to be in iOS store the Apple Developer License is $99 a year.  **Time:** Due to homogenous nature of mac hardware testing time should be minimal.  **Expertise:** Swift, HTML5, CSS3 are the important. | **Development Considerations:**  Linux users have various distributions, screen resolutions and browsers. The web app must be compatible with all of these.  **Cost:** Open source so none to minimal licensing.  **Time:** Time considerations would focus on cross browser compatibility.  **Expertise:** JavaScript, HTML5, CSS3 is key but the development team must be familiar with Linux in general for testing. | **Development Considerations:**  Focus should be on compatibility across browsers. Platform differences minimal (windows 7,10,11 etc.)  **Cost:** After server-side licensing no addition cost.  **Time:** Time considerations would focus on cross browser compatibility.  **Expertise:** JavaScript, HTML5, CSS3 are the important skills. | **Development Considerations:**  Mobile development should focus on adapting between different screen sizes and orientations, as well as touch inputs.  **Cost:** Minimal cost for web client-based applications, only a minor consideration if native apps are developed for platforms on Play Store or App Store (aka not prohibitive).  **Time:** Time considerations would focus on mobile web development of the game, along with touch controls. Other phone specific considerations would be required (notifications, getting calls, etc.)  **Expertise:** Swift, JavaScript, HTML5, and a strong mobile development knowledge would be required. |
| **Development Tools** | **Languages:**  HTML5 CSS3, Swift  **IDEs and Tools:**  Xcode,  Visual Studio Code  **Impact:**  Team needs knowledge of creating for Safari as well as Swift proficiency.  **License Costs:**  Moderate, hardware is biggest investment in an iOS project. | **Languages:**  JavaScript, HTML5  CSS3, Python, Java.  **IDEs and Tools:**  Eclipse, Vim, Visual Studio Code  **Impact:**  Team needs Linux knowledge and cross platform knowledge as most players probably won’t be on Linux Systems.  **License Costs:**  Low due to so much being open-source software. | **Languages:**  JavaScript, HTML5  CSS3, Python, Java.  **IDEs and Tools:**  Visual Studio, Eclipse, PyCharm  **Impact:**  Team needs knowledge of .Net and Microsoft Server environments.  **License Costs:**  Middle to High for Microsoft Licenses. | **Languages:**  Swift (iOS) or Java (Android)  **IDEs and Tools:**  Xcode, Android Studio  **Impact:**  Team needs separate skill sets for iOS or Android or someone who is a cross-platform expert.  **License Costs:**  low, some if creating natively. |

## 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**: My recommendation for operating platforms would be on the Server-side to utilize Linux, while on the Client-side focus on development for windows browsers. The reason behind this is that while there is some extra knowledge required, a Linux system is relatively inexpensive to set up and maintain while also providing a powerhouse of features and support. Development of the client’s side should start, in my opinion, with Windows browsers. This is due to having broad reach and easier implementation and testing as well as basic compatibility with iOS via HTML5 and CSS3. From there, The Gaming Room can branch out into other client platforms such as Android, and mobile iOS.
2. **Operating Systems Architectures**: Linux is a highly modular operating platform that is already used throughout the world for server systems. Because of its modularity, only the functions necessary for running the server can be installed, allowing for an increased layer of security and performance. Linux is also highly scalable, allowing for increased workloads by utilizing hardware like multi-core processors efficiently as well as supporting additional servers later to cluster the workload. The monolithic kernel design also keeps performance streamline by having a single address space where all services run in the same memory space. This works by loading modules directly into the kernel. The end result is a customized and speedy server. On top of that it is open source so it doesn’t face the same licensing issues that Windows or iOS would have when considering the cost.

The architecture behind the client is more familiar to a vast number of people with a windows PC. It is a mostly preconfigured operating system that also supports multi-core processors, has less modularity than Linux (but more than iOS), can handle virtual memory, if need be, and a slew of other things to improve performance. Browsers operate in the user space of Window’s hybrid kernel and the browser itself provides a nice sandbox to help isolate the application from the rest of the system. Since the workload for a browser game is minimal versus a server, the main architecture is less of a factor than the server.

1. **Storage Management**: Linux supports scalability in a variety of ways; however, I believe most of the storage should be done using a cloud service to deliver the bulk of the resources. This will offset the cost of having to store and distribute the 1.6gb set of 8mb images required for the game to the users. For the Linux server specifically, it is compatible with a variety of storage options. Using a redundant array of independent disk system (RAID) allows for increased performance as well as reliability by having redundant data in place automatically. On top of that if the game requires upward scaling, multiple servers can be clustered together to provide additional storage options.

Storage on the client side should be minimal. It will be a browser-based game, so some assets can be cached. I recommend using a Least Recently Used policy with a hard cap on it so a user’s system does not grow too big, but they can still quickly load a recent asset they used. On top of this I recommend storing these photos in a lower quality thumbnail as well to send out as a temporary image until the 8MB one load.

1. **Memory Management**: Linux has a variety of customizable memory management techniques to improve performance. Virtual Memory allows the system to write to disk space if handling large numbers of concurrent sessions and extra memory is needed. Paging is also a technique that efficiently manages memory by dividing it up into grouped segments which prevents fragmentation and makes accessing related memory faster. Linux can also utilize shared memory, which allows for multiple processes to pull off the same location in memory. This is especially useful in an online game where the same variables may be reused frequently. Proper implementation of these methods, however, will require a certain level of knowledge going into the creation of the system. Memory on the client side should not be an issue for a browser game unless the player is operating on a very old system.
2. **Distributed Systems and Networks**: Since the system is to eventually be on multiple systems and devices several early adoptions would go a long way toward this goal. First utilizing the JSON format for sending packet information would make data easily transferable between systems. Since JSON is readable in all modern languages, no matter where The Game Room wants to deploy next the data being sent form the servers will not need to be modified. Next using a common language such as JavaScript or HTML5 would allow the game to be operable on any modern browser regardless of the system. As for network consideration, the primary concerns are going to be latency and security. Latency can be helped by Load Balancing between available servers to ensure network traffic is not overloading one node. Also, utilization of cloud servers to deliver game images will reduce network overhead greatly and help prevent outages. Security of the system will be helped by firewalls and encryption as stated in the next section.
3. **Security**: Security for an online game is a must considering bad actors could try to cheat their way to the top which would push out legitimate players. The first stage of security would be encryption. This should both at rest by utilizing encryption on the server to safeguard user and game information, as well as in transit by utilizing HTTPS to encrypt data between server and client. Secure log in is also something that needs to be addressed, multi-factor authorization is an option utilizing an authenticator like Google or Microsoft on your phone to serve as an additional layer to access your account. Finally, a properly configured firewall for the server would help ensure traffic was properly blocked unless specifically authorized.