

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)

[**Design Constraints 3**](#_Toc115077322)

[**Domain Model 3**](#_Toc115077324)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/23/2025 | Brycen McEuen | Original document |

## [Executive Summary](#_sbfa50wo7nsh)

As a technology consultant for *Creative Technology Solutions*, I’ve been tasked with helping *The Gaming Room* develop a web-based game capable of serving multiple platforms, based on their current game *Draw It or Lose It,* which is only available on Android at the moment. The game should have multiple teams, with each team having multiple members. There will be four rounds lasting a minute each, and the objective is to guess the puzzle within that timeframe. If a team fails to guess before the time is up, another team will have a chance to guess the puzzle.

## [Design Constraints](#_2et92p0)

* Developing a web-based application might introduce latency and performance problems, decreasing the product's overall quality. The game should be designed to minimize the effects of latency by using efficient networking protocols and program optimization.
* The game should be able to handle a large number of users at the same time, without sacrificing performance. To effectively scale, the application must run on reliable servers to ensure high performance.
* The game must be cross-platform, meaning it should be compatible with many different devices. This means building the system with a wide variety of technologies in mind, making sure that it is tested extensively to ensure smooth operation on all platforms.

## [Domain Model](#_8h2ehzxfam4o)

In this UML Diagram, the ProgramDriver class holds the main, which is used to actually run the program. The ProgramDriver class also uses the SingletonTester class, to test that the program is working as intended since this program design will be based on a singleton pattern. Now, the Entity class is what we will use to uniquely identify each individual game, team, and player. Each entity will have its own unique ID and name so that a player can’t choose a name that is already in use. We also have three separate classes that inherit from the Entity class to function, those being the Game, Team, and Player classes. The Game class will have a list of the teams that are actively playing the game, as well as the ability to add teams if needed. The Team class has a list of players that are actively playing the game, as well as the ability to add players if needed, and the Player class simply holds each player’s ID and name. Lastly, the GameService class is the largest and most complicated class in this program. The GameService class is basically where the games will be made since that’s where the instance of the game is actually created. This class holds things such as game IDs, player IDs, and team IDs, as well as being able to create more games when needed.

**"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.**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macs have a reliable and secure operating system, native development tools, and built-in servers. While they’re good for development purposes, they sometimes lack when used as hosts because of their limited server support and high maintenance cost. Most hosting could be done using open-source platforms, meaning no licensing costs. | Linux is widely used for hosting web-based applications, mostly due to it being free and open source. They have plenty of server support, high performance, stable security, and a large support community. Its drawbacks are that it has a steep learning curve and less native software support. Most popular server options are free and open source, meaning there would be no licensing costs involved. | Windows is easy to learn, user-friendly, and supports a wide variety of software. It's frequently used in enterprise environments for its ease of use and community support, but it does suffer from high resource usage and the fact that it’s less development-friendly compared to Mac or Linux. Usually involves paying for licenses for both Windows Server and whatever other software you choose to use. | Mobile devices aren’t generally used as servers, but they technically can be. Their advantages are that they’re portable and are good for small demos, but unfortunately, they lack in all other areas, such as performance, security, stability, and tools/integration. Hosting on a mobile device most likely would not involve any licensing costs due to the fact that the more popular options for mobile server hosting are open source and free. |
| **Client Side** | When it comes to Mac devices, this game would most likely run on Safari since it is the most widely used web browser, which means it needs to be tested specifically to work run smoothly on Safari. This could mean making sure it supports modern web standards, making sure the gui interface works as intended, and making sure it’s optimized to run smoothly. | Linux users would most likely play this game through either Firefox or Google Chrome, as those are the most popular choices on Linux devices. Once again, make sure its tested with those browsers to ensure functionality with GUI, performance, and optimization. | Even though Microsoft Edge is the default web browser on Windows, Google Chrome is still the most popular, so that should be the primary focus. Just like with Mac and Linux, it should be tested to ensure it works specifically with Google Chrome. | Mobile devices generally use either Safari or Google Chrome, depending on whether it's Android or iOS, so those should be the focal points for development. Again, make sure it runs correctly on those browsers and that the performance is optimal, especially because mobile devices are usually weaker in terms of performance than computers. |
| **Development Tools** | Developing this type of application for Mac would include using front-end programming languages such as HTML, CSS, or JavaScript, an IDE such as Visual Studio Code or IntelliJ IDEA, and, of course, a language like Python or Java for back-end development. It would also need a framework such as React or Angular. This could be done with just one team of developers. | Developing this type of application for Linux would include using front-end programming languages such as HTML, CSS, or JavaScript, an IDE such as Visual Studio Code or IntelliJ IDEA, and, of course, a language like Python or Java for back-end development. It would also need a framework such as React or Angular. This could be done with just one team of developers. | Developing this type of application for Windows would include using front-end programming languages such as HTML, CSS, or JavaScript, an IDE such as Visual Studio **(natively** **supported)** or IntelliJ IDEA, and, of course, a language like Python or Java for back-end development. It would also need a framework such as React or Angular. This could be done with just one team of developers. | Developing this type of application for mobile devices would involve a little more work. For iOS devices, this might include using a programming language like Swift, an IDE such as Xcode, and a framework like React Native or Flutter. For Android, it might include languages like Java or Kotlin, an IDE such as Android Studio, and React Native or Flutter.  Depending on your staffing situation, this might require multiple teams due to the fact that the iOS and Android platforms would have to be developed separately. |

## 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 choosing Linux to be the primary operating platform for *Draw it or Lose it*, mainly because of its stability, security, performance, large community, and cost-effectiveness. It’s known for being a reliable and ideal choice for hosting and managing web-based servers, and is generally the preferred option for most companies.
2. **Operating Systems Architectures**: The best thing about Linux is how flexible it is, which is why it has gotten incredibly popular for server hosting and software development in general. Linux uses a multi-layered system composed of different elements such as the Kernel, System Library, System, Hardware Layer, and Shell functions. They all come together to form the OS.
3. **Storage Management**: I would recommend using a system like ZFS for storage management, as it is ideal for storing game data when considering its built-in compression and snapshots, which would be perfect for *Draw it or Lose it*, since it will need to have lots of pictures available.
4. **Memory Management**: Linux uses a wide variety of advanced memory management techniques to ensure efficient performance in web web-based game environment. A few examples include virtual memory management, memory caching and buffering, and memory overcommit handling. Virtual Memory Management is effective because it ensures that each player’s memory usage is isolated, preventing their memory allocation from affecting others. Caching and buffering are helpful because they cache data in memory to minimize I/O operations, which reduces latency and ensures quicker load times. Finally, memory overcommit handling is useful because it prevents crashes due to memory exhaustion.
5. **Distributed Systems and Networks**: To allow for communication between different devices on different networks, we need to ensure the use of HTTP API’s (which we already have implemented), and proper communication protocols. HTTP API’s allow the client to send and receive information from the server in order for the game to function. Communication protocols are important as well because they provide a way to process multiplayer interaction. Some dependencies to think about might include server reliability, database reliability, network reliability, and external issues such as outages. It’s important to make sure each of these components has backup devices to help in case of any failures that may occur.
6. **Security**: Security is always a big concern with online applications, but luckily for us, Linux offers a robust set of security mechanisms that can help keep people's data safe and secure from potential threats. For example, Linux offers many advanced safety features such as data encryption, database encryption, access control/permissions, secure network protocols, and monitoring/incident response.

**Sources**

*Best Server OS for Your Website: Linux or Windows?* (2023, April 20). Liquid Web. <https://www.liquidweb.com/blog/best-operating-system-for-web-hosting/>

*Linux Architecture - Detailed Explanation*. (2022, June 10). InterviewBit. <https://www.interviewbit.com/blog/linux-architecture/>

*What is ZFS? - Definition from WhatIs.com*. (n.d.). SearchStorage. <https://www.techtarget.com/searchstorage/definition/ZFS>

*Types of Network Protocols and Their Uses*. (2021, November 22). GeeksforGeeks. [https://www.geeksforgeeks.org/types-of-network-protocols-and-their-uses/#](https://www.geeksforgeeks.org/types-of-network-protocols-and-their-uses/)

‌

‌

‌

‌