

Draw It or Lose It – Web-Based Version

# **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 | 05/15/2023 | Joseph J. Les | Initial Doc review |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room tasked us to turn their Android game “Draw it or Lose it”, turn it into cross-platform web-based game. Our goal is to create a robust, responsive game and will be utilizing a comprehensive shape library for all players to interact with the game easily, irrespective of their device or browser. We were also asked to integrate a feature to allow more than one team to play at a time, leading us to develop an efficient back-end system capable of managing multiple game sessions and player interactions.

## Requirements

The existing version of "Draw it or Lose it" game is presently confined to Android devices, which restricts its accessibility. To extend its reach, there's a compelling need to transform the game into a cross-platform application that supports all devices, thus increasing its potential user base. Additionally, to accommodate a potentially higher load of concurrent games and ensure optimal performance, we need to design the game to be playable on either client servers or advanced cloud services, depending on the preferred infrastructure model. Lastly, recognizing that the digital landscape is constantly evolving, we will be committed to regular maintenance and updates. This will involve adapting to new technological trends, rectifying any bugs, and implementing improvements to ensure the game's performance remains high-quality, consistent, and in line with user expectations.

## [Design Constraints](#_2et92p0)

In developing a web-based application, we must consider critical factors such as latency, cross-platform capabilities, security, and scalability. These considerations will inevitably impact the application development process in the following ways:

• Development time and costs will increase, as developers will need to utilize more advanced technologies and tools to meet these requirements.

• The range of team skills required will expand significantly. Team members will need to understand networking, security, cross-platform development, and UI design.

• Regular maintenance will be required for the game to stay updated with continuous changes in browser technology and device updates.

## [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 following UML Diagram incorporates the principles of Object-Oriented Programming (OOP). It employs inheritance, enabling the child classes - Game, Team, and Player - to inherit from the parent class, Entity. Within these classes, encapsulation is applied to conceal the internal operational details of the objects. This is achieved by designating certain methods as private. Further, the ProgramDriver and SingleTester are depicted separately from the rest of the diagram, signifying that these functions do not interact directly with the classes, but rather act upon them.

**"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** | Mac is renowned for its stability, a characteristic that is critical for server applications. As Mac OS is Unix-based, it lends itself to seamless integration with server technologies such as PHP, Rails, Apache, and Ruby, all of which can be set up without encountering major configuration problems. | Linux provides an excellent platform for hosting websites due to its superior maintenance capabilities and robust security features. Its simplicity ensures easy navigation and updating. Moreover, being an open-source operating system, Linux allows for cost-effective solutions and extensive customization. | Unlike Linux, Windows servers provide a graphical user interface (GUI) for easier interaction. Furthermore, Microsoft servers are designed to integrate seamlessly with other Microsoft products, and the support offered is comprehensive, including regular updates and security patches. However, there are some drawbacks. Notably, users must pay a licensing fee, and performance may not be as optimal as that offered by Linux. | Mobile devices are primarily used for client devices and not as servers. |
| **Client Side** | While Mac hardware and software are typically more expensive compared to other platforms, they deliver a unique user experience and high-quality performance. Developers will be face will developing and testing cross-browser capabilities. It might be easier to maintain optimization between Apple ecosystem, it will take more developers to be able to develop across all systems. | Designing a web-based game for the client side on Linux will require testing across multiple client configurations. This will inevitably lead to an increase in testing time as well as additional costs for hardware or virtual machines. Furthermore, substantial work will be required on the user interface to ensure compatibility with various Linux distributions. | Given Microsoft's widespread familiarity and establishment among users, clients often find it easy to use, and it boasts extensive support for multiple browsers. However, Microsoft is frequently targeted by malware, which presents a significant weakness in terms of security. | Mobile devices provide the advantage of a wide range of screen sizes, touch screens and portability. Leading to advantages of ease of use for touch screens, location services can be used within the game and web applications can take advantage of camera, accelerometer and more. Weakness are performance limitations, network dependency since they are wireless and having to test on a variety of devices. |
| **Development Tools** | There are several programming languages that can be effectively utilized on Mac OS. For web development, a few notable ones include JavaScript, Python, Ruby, and Swift. When paired with Integrated Development Environments (IDEs) like Xcode, Apple's official IDE, or Visual Studio Code, the performance of these languages can be maximized. | Linux offers a broad selection of programming languages to choose from. For instance, Python, Perl, PHP, Java, and C/C++ can be utilized. In addition, HTML and CSS are also available for web development. Depending on the chosen GUI, Linux supports a range of Integrated Development Environments (IDEs), including but not limited to Eclipse, NetBeans, IntelliJ, and Atom. | Some of the languages that work best with the Microsoft ecosystem include C#, JavaScript/TypeScript (which is excellent for web development), SQL, and Python. Although Microsoft supports several IDEs they have developed their own – Visual Studio Code (VS Code). This IDE offers a wide range of extensions for customization and performance optimization. | Even though it is possible development on mobile devices, it is extremely limited due to hardware constraint, screen size and lack of tooling. |

## Recommendations

1. Operating Platform: I recommend that The Gaming Room adopt a Linux-based platform for their game's operation and functionality. Linux offers an excellent server platform thanks to its open-source nature, robust security, user-friendly design with extensive documentation, and superior performance. Its stability and reliability will enable the game to operate 24/7.
2. Operating Systems Architectures: Client-Server Architecture with a two-tier layout would be the recommended strategy for The Game Room. This allows them to have their customer facing servers running the UI, then having their client servers make the request to the database. Having this physically separated will increase functionality and security.
3. Storage Management: Cloud providers will be the best solution since they are able to host the game on each coast, reducing latency for the customer. This strategy will also make it easy to have redundancies as well as expanding their storage as needed. For long-term storage of player information, cloud providers offer a cheaper solution that does not require weekly or monthly access. This would be long term strategy as their fan base grows.
4. Memory Management: The EXT4 file system's journaling feature provides protection against data corruption, reduces file fragmentation with delayed allocation, and offers improved read and write performance. It also supports large individual files, which will be beneficial for games with a larger number of teams or players.
5. Distributed Systems and Networks: By leveraging the capabilities of cloud technology, I recommend placing servers in at least two different regions using a cloud provider. This strategy will facilitate redundancy and load balancing to enhance performance and reduce latency. Linux supports file systems like GlusterFS or NFS, allowing file sharing among multiple servers.
6. Security: Ensuring security while transmitting data across various platforms is vital for user confidence. We can achieve this by encrypting data both at rest and in transit. For in-transit data, secure protocols such as HTTPS or SSH should be used exclusively. Requiring users to set up two-factor authentication will further secure user accounts. Lastly, we will adhere to security standards and frameworks like ISO 27001. Lastly, by using two tier client server architectures, the customers data and infrastructure code will never be expose to the internet.