

Draw It or Lose It Web Development

# **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 | 1/27/2024 | Ollie Simmons | Established executive summary, requirements, design constraints, and description of the UML diagram |
| 1.1 | 1/28/2024 | Ollie Simmons | Evaluated development requirements for the different operating systems and chose Windows as the OS of choice for the project based on its advantages and negligible disadvantages |
| 1.2 | 2/12/2024 | Ollie Simmons | Expanded upon storage management and data management, the evaluation was already mostly complete |
| 1.3 | 2/25/2024 | Ollie Simmons | Expanded on the descriptions for the recommendations regarding Windows as the operating system of choice |

**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 has a game called Draw It or Lose It that is only available on an Android app. They want a web-based version of this game that can be made available on a variety of platforms, but do not know how to set up the web-based environment. Ideally, the game would need to be accessible on all browsers such as Microsoft Edge, Google Chrome, and Mozilla Firefox, and it would require cross-platform compatibility between all platforms, including its already existing version on the Android app. Its UI would need to be built to be easily interactable and accessible on both touchscreens and desktop environments, not to mention scalable on a variety of different screen sizes. Assuming user data is to be stored for returning users, a system would need to be built to store that data and retrieve it from all of these different platforms. A frontend and backend system would need to be put in place that works fluidly and responsively across all these platforms, meaning we will have a lot of testing to do. Security measures will also have to be put into place to protect user data and guard against common web vulnerabilities that could negatively impact user experience.

## Requirements

*As far as business requirements go, the game must support having a single game instance, but with multiple teams and multiple players within those teams. There must also be efficient user data management that is protected from duplicate identifiers or names for the games, teams, and players. For the technical requirements, singleton design patterns will need to be implemented to ensure only one instance of the game be active in memory at any time. In addition, iterator patterns will be ideal in ensuring no duplicate identifiers or names for the games, teams and players. The game must also allow for users to interact with the system in real time as the drawings are generated. Other requirements exist that can be described as both business and technical: the cross-platform capability and UI scalability will be a necessity in order to reach as many users as possible and security features will need to be put in place to ensure the safety of user data and guard against those who may attempt to tamper with the game environment.*

## [Design Constraints](#_2et92p0)

* The game must be web-based and available on a wide variety of platforms. Cross-platform compatibility will almost certainly require extensive testing across all devices and environments to ensure a seamless user experience. Methods of UI scalability will need to be investigated to also keep to this.
* The must only be one game instance running at a time in memory. This is no longer going to be enforced on just The Gaming Room’s Android app; it will need to be regulated across every platform and environment the game will be playable on. Singleton patterns will be a must in the system that will be put in place to manage this.
* Multiple teams and multiple players will need to be supported within the game instance. This calls for a robust system that will go to ensuring unique identifiers and names for these teams and players to prevent conflicts within the game. Effective iterator patterns will need to be put in place to ensure the uniqueness of user data.
* The environment must respond to all these users and their input within real-time as the drawings are generated. Security measures will of course need to be put in place to prevent user tampering within this shared environment. It must also support scalability as the number of teams and players grow.

## [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 contains seven key classes: ProgramDriver, GameService, Entity, Game, Team, Player and SingletonTester. The ProgramDriver class runs the main application and uses the SingletonTester class for testing purposes. The Game, Team, and Player classes all inherit their identifier and name values from the Entity class as well as its accessor methods, an example of the OOP principle of “inheritance” which helps to avoid manually distributing the values and methods across the classes and allows for possible expansion in the future. The Game class allows for the creation of unique Team objects and stores a list of all the currently existing ones; similarly, the Team class allows for the creation of unique player objects, indeed storing a list of all the existing players. Each single game can have many unique teams and each single team can have many unique players. With each entity, game, team, and player object having unique identifiers and names, the goal of The Gaming Room having a robust system of keeping all game, team, and player identifiers and names unique is fulfilled. Furthermore, this is a prime example of the OOP principle of “polymorphism” which allows the same object to appear in a wide variety of forms. There are also examples of the OOP principle of “encapsulation” here which allows for security over data when needed: the GameService class stores the singular running instance of the GameInstance object utilizing a singleton method which takes advantage of private variables and a private method as well as a public accessor method. In addition, the classes store private arrays containing the objects they create, data of which can only be retrieved via their getter methods. All of this goes hand-in-hand with The Game Room’s constraint of ensuring the security of game data. Lending further to the “polymorphism” method, this single running instance of GameService can hold a great many individual games. The OOP principle of “portability” is also present here as every one of these classes are structured to be able to be ported to plenty of other operating systems without requiring any major rework.

**"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** | The Mac operating system is less commonly used than Windows, but more commonly used than Linux. It doesn’t have as much versatility as Linux or Windows, but there are plenty of people who have successfully hosted systems on this operating system. It is by no means a niche environment to build the server-side in. | Linux is built to be much more accessible for professional programmers which could lend well to the development of this project, but it certainly has the least amount of familiarity. This shouldn’t really impede the server-side hosting of it as virtually all web-based or cloud-based hosting systems are compatible with almost all of the major operating systems. | As far as weaknesses go, there are virtually none. Windows is usually the industry go-to for web development due to its familiarity and versatility. Development labor will be plentiful here. | Building and hosting a web-based software upon a mobile device will generally be much more restrictive. It isn’t even very commonly heard of for such systems to be developed this way. Mobile devices of course offer the most simplicity but the versatility pales in comparison to what can be accomplished via mouse and keyboard. |
| **Client Side** | Cost will be a bit higher for a Mac development environment than that of Windows, though higher than that of Linux or mobile devices. Its slight weakness in versatility may also cost time. | Linux will be better built for the programming aspect of the project, but there just aren’t as many programmers versed in Linux as systems such as Windows and Mac. This will inevitably lead to more cost and time, but the end result of the project should indeed fulfill The Gaming Room’s expectations. | There are more programmers available for Windows client-side building than on any other operating system. It also has plenty of versatility and capability, so the project should be less costly or time-consuming than on any other OS as well as fulfill the design constraints of The Gaming Room. | Development on a mobile environment will be the most time-consuming due to its lack of versatility. This could lead to be the most time-consuming and costly result of the client-sided aspect of the game. |
| **Development Tools** | Mac utilizes MacOS as its operating system. While it supports virtually all browsers, it is best suited for Safari. Its most common programming language is Swift. | Linux’s environment is best suited for professional programmers. It can support a wide variety of programming languages. | C and Java are common on the Windows OS, but it supports almost all programming languages. It prefers for its users to use the Edge browser, but almost all browsers including Firefox and Chrome are available and work seamlessly. | This depends on the OS of the mobile device itself. IOS devices will commonly use the Swift programming language. |

## 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**: Due to Windows’ familiarity and versatility, this is my go-to for the operating system of choice. It will attract the most amount of developmental labor out of the other systems, even despite it not being as versatile programming-wise as a system such as Linux, offering extensive access to major IDEs which enhances flexibility. It can also be easily used to expand the scope of Draw It or Lose It to be cross-platform.
2. **Operating Systems Architectures**: Windows can be used for almost any programming language and can support almost all major server-side and client-side hosting environments. This grants flexibility in the development process among the programmers. It can also support almost all major IDEs for development such as Eclipse. Languages like Python and Java can be easily used for the web development aspect of the project as well as almost any other language, giving a wide variety of options for almost any developer.
3. **Storage Management**: Windows is compatible with a wide variety of hardware storage systems and cloud-based storage systems. It’s easily expandable and the OS is equipped with a robust storage management system that is easy to navigate. This lends to the storage management aspect of the project to be easily scalable and flexible.
4. **Memory Management**: Almost all operating systems are built to seamlessly manage memory. The Windows platform is no exception, ensuring sophisticated memory allocation and resource management. At the end of the day though, any one of the big three operating systems can be used to fulfill this purpose as they’re all built to near-flawlessly manage memory. Proper memory management is more important than it has ever been in today’s world of computing.
5. **Distributed Systems and Networks**: Almost all web-based and cloud-based platforms are compatible with Windows and can be used to communicate with other operating systems. Languages like Java and Python can be ported to almost every existing OS for testing and development, further ensuring cross-platform compatibility and allowing for seamless collaboration and connectivity.
6. **Security**: Windows is sometimes scrutinized for its lack of transparency about how it handles user data. It definitely does not have as much security over it as something like Linux, but it boasts built-in capabilities for data encryption and user authentication. It contains all the necessary tools to implement best practice security measures to prevent from duplicate user identifiers and names as well as user-tampering to ensure a seamless user experience.