

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 07/16/2025 | Bwezani Mukuka | Initial draft of software design document for the Draw It or Lose It game. |
| 1.1 | 07/29/2025 | Bwezani Mukuka | Reviewed the evaluations section, revised and expanded on the different platforms. |
| 1.2 | 08/12/2025 | Bwezani Mukuka | Improve and revise the recommendation 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 paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to make a web-based version of their android game ‘Draw It or Lose It’ to work on multiple platforms. The game is played by multiple teams of different sizes that compete for 4 rounds and last for 1 minute. The major challenge is to ensure that each game, team, and player will have a unique name and only one game can be active at any time.

## Requirements

The following requirements will guide the design of Draw It or Lose It:

* Game and team names must be unique. It should also allow the user to check if a name has been taken.
* Each team must have multiple players.
* The platform must support web-based, cross-platform functionality.
* Only one instance of the game must be in memory at a time.

## [Design Constraints](#_2et92p0)

Some constraints we may come across:

* The application must run on multiple devices and browsers. Platforms such as Java, HTML5, and JavaScript should be compatible.
* The application must be built with future updates in mind.
* Set a name check to verify that no names match before allowing a game/team to be created by an end user.

## [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 has four parts that are central to this game: Game, Team, Player, and Entity. The Entity portion is a base for basic information like a name and ID number. Game, Team, and Player receive this basic information from the Entity so that we do not duplicate code unnecessarily – this is an example of inheritance, or if one class can share things with other classes. This example also includes encapsulation, which means the classes hold their own separate and protected pieces of data, and have a specific and organized purpose with that data. The GameService inherits from the Entity and helps to create and manage games using the Singleton Pattern, this means that only one game service will run at a time. The classes also reimplement the Iterator Pattern by looping through lists of games, teams, or players to avoid duplicate names. The overall design allows for a neat organized town which can be held at one instance and extended when necessary.

**"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 can run a web app but it is generally not used as a server. Mac is usually more expensive than the other options and does not include a lot of server tools to run large websites. | Linux is best suited to host websites. Linux is free, secure, stable, and is friendly when working with common server tools. | Windows can host websites using certain tools, such as IIS. Windows has an easy to use interface but costs money and continuity security problems or vulnerabilities. | Phones and tablets can't be used as servers because they don't have sufficient power, and most devices aren't designed to run a website to be shared with many other users. |
| **Client Side** | In order to get the app to work on Mac, developers have to use a Mac computer and must comply with Apple's design guidelines. It takes more time and money. | For Linux, the app should work fine in the Chrome and Firefox browsers. However, it requires a little less user support, so they could be more testing that is needed. | Windows is the most common for users. The app should work in Edge and Chrome, and the developers will have to make sure it works on all Windows versions. | To support iPhone and Android at the same time, you build two separate apps. This increases time and complexity to you, but you have no choice if you want to cross-platform. |
| **Development Tools** | You can use tools like Eclipse, IntelliJ, and Xcode (which is required to build iPhone apps). You must have a Mac to build iOS apps. | Works well with free tools like Eclipse, NetBeans, or Visual Studio Code. It's a good option for developers and it is free. | You're able to use tools like Visual Studio, IntelliJ, or Eclipse. Visual Studio works well with Microsoft tools like Azure, but may cost some money. | You're going to need Android Studio and Java for Android. For iPhone, you will use Xcode and Swift. You may also use React Native to build one app for both. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* **Operating Platform**: With expansion plans for Draw It or Lose It in mind, the recommended Operating Platform is Windows Server. Windows Server offers excellent scalability, robust developer support, and seamless compatibility with Azure cloud services, making it a great choice. Windows, as an operating system, supports enterprise applications very well. Windows supports standard deployment tools and development environments, which makes development and implementation easier. With this operating platform, you can also utilize a layered architecture to separate responsibilities, making it easier to maintain as the application expands.
* **Operating Systems Architectures**: The system uses a simple three-part design. First is the part you interact with—the web and mobile screens that make it easy to sign in, play, and see results. Second is the game’s “brain,” which applies the rules, keeps score, and manages each player’s session behind the scenes. The third is the data layer. It securely stores and retrieves player and team information, ensuring that progress and records are always available. Keeping these parts distinct makes the system easier to change and test. We can update the interface without affecting the rules, adjust the rules without compromising the data, and improve the database without impacting the rest. The result is a more reliable, flexible system where new features can be added faster and with fewer problems.
* **Storage Management**: A hybrid storage option is the recommended approach. Structured information (users, teams, game history, etc.) will be managed using an interactive database like SQL Server. As the application expands, only a subset of services and layers can be extended if necessary due to the vertical/horizontal scalability of the cloud options (backend). There are also more options to manage user details, player profiles, and content information using document storage options without unnecessary overhead.
* **Memory Management**: Windows Server features both physical and virtual memory management that can be used to optimize performance. Virtual memory enables processes to operate smoothly even when physical RAM is limited. Paging also ensures that data is isolated and protected from other processes. The runtime environment can manage the automatic garbage collection and reclaim memory that was not freed during use, preventing memory leaks that can happen during long-lived sessions. Additionally, caching frequently and recently used assets adheres to the principles of the Flyweight pattern, which teaches us that we can share an object if it is not duplicated, allowing for less memory to be used and subsequently faster responses during peak load times.
* **Distributed Systems and Networks**: The game will transfer information between players and to the server over the Internet. We will use the current secure websites (HTTPS) to perform this safely; security and data privacy is very important. Part of how Windows itself can create, secure, and distribute traffic across multiple servers is that if the player base is large and they all want to enjoy the game at the same time. These tools also help track any problems, so they can be fixed quickly.
* **Security**: Security controls will be established throughout the system, including end-to-end encryption for all communications. Sensitive user data, such as passwords, will be hashed. Role-based access control will restrict administrative permissions, and multi-factor authentication will provide an additional layer of protection for accounts. Firewalls and authentication services set at the server level will protect system resources. This significantly reduces the potential impacts of unauthorized intrusions. By keeping the platform patched, monitored, and compliant with established industry standards, it will be protected from multiple evolving threats.