

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 | 11/12/2023 | ML | Added Executive Summary, Requirements, Design Constraints, System Architecture View, Domain Model, Evaluation |
| 1.2 | 12/10/2023 | ML | Added Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to transform its existing game for the Android platform, Draw It or Lose It, into a web-based platform to expand its accessibility across multiple devices. They wish to design a scalable system that allows for seamless gameplay with multiple teams and players, with unique team, game, and player IDs.

## Requirements

1. The game supports multiple teams
2. Each team contains multiple players
3. Game and team name must be unique
4. Only one instance of the game in memory at a time

## [Design Constraints](#_2et92p0)

* Cross-Platform Development - This online multiplayer game must play on all major platforms – Mac OS, Windows, and Linux, and mobile platforms.
* Consistent User Experience – This app must work seamlessly between mobile and web-based versions to grow the userbase and maintain the user experience gamers have come to expect from Draw It or Lose It.

## [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 provided code was refactored so that Game, Team, and Player inherit from an Entity class. This was done to improve the code’s organization and decrease the redundancy of having id and name fields in three different classes. By introducing an Entity class that multiple classes inherit from, we introduce the object-oriented programming concepts of inheritance, encapsulation, and abstraction. Additionally, it ensures each game, team, and player has a unique ID to avoid issues in future iterations 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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS server was discontinued, making Mac a poor choice to host a server. Additionally, Mac server solutions might require proprietary Apple hardware. | Open-source platform, secure, and versatile. Ideal choice for server, due to their stability, scalability, and cost-effectiveness. Minimal licensing costs. | Second most ideal choice for hosting a server. Large market share and many options, but involves licensing fees. | Limited processing power means mobile devices are a poor choice to broadcast a server from. |
| **Client Side** | Cost is expensive – mostly from purchasing Apple hardware, but time and expertise are moderate. | Very cost-effective as many tools for Linux are open source. Expertise and time are a bit higher. | Expensive, but less so than Mac development. Windows development tools may require special licenses. Efficient development means less time taken. | Testing on various devices and developer tools can become costly. Mobile development requires Java and Swift, and can take longer is creating separate codebases for iOS and Android. |
| **Development Tools** | HTML, CSS, Javascript, SQL for databases, Visual Studio, Xcode | HTML, CSS, Javascript, Python, Java, C++, SQL for databases, Visual Studio, Cmake | HTML, CSS, Javascript, C#, .NET, SQL for databases, Visual Studio | Node.js, Java, Python, SQL for databases, Android Studio. Swift and Xcode for iOS. |

## Recommendations

Here are the following recommendations we have for Draw It or Lose It:

1. **Operating Platform**:
   1. Linux is the recommended choice for server architecture, as it is open-source, free to use, and widely used and documented in the game development world. Additionally, there are minimal licensing costs. This is in line with The Gaming Room’s budget considerations.
   2. The game itself should be available on Android, iOS, and Windows/macOS/Linux through a web browser version. Developing a browser-based version of the game is easier to implement than designing discrete programs for both Windows, Mac, and Linux operating systems. For a simple party game, users will likely not want to download anything new to their personal computers and will appreciate the accessibility of a browser-based version.
2. **Operating Systems Architectures**: A web-based architecture such as HTML5 is ideal to create a secure game and the responsive user interface needed to play Draw It or Lose It on multiple platforms.
3. **Storage Management**: While on-premises servers that run the game’s infrastructure are possible and provide a good deal of independence, it is a costly and time-consuming process. Cloud-based storage solutions (Amazon AWS, Google Cloud) will allow Draw It or Lose It to access game data across multiple platforms and will make the game available to a worldwide userbase. Google Cloud contains features that are recommended for scalability such as Cloud Spanner for relational databases that can scale for performance as the game and userbase grows.
4. **Memory Management**: Modern web browsers have good memory management and smart tools, but performance optimization will still be important. Consider caching (storing frequently used data in memory) to avoid needing to constantly retrieve the information and dynamic loading to access game files.
5. **Distributed Systems and Networks**: Consider a third party such as Cloudflare to accelerate traffic and boost performance with an Anycast network.
6. **Security**: With designing for a browser-based game app, make sure the user is prompted to be on a modern, secure version of their web browser. Consider restricting it to the most recent Firefox, Chrome, Opera, Safari, and Edge versions. Additionally, implementing measures to encrypt data transmission is important. Consider HTTPS – the secure version of HTTP, which sends data between web browsers and a website.