

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 11/11/2024 | Janai Williams | Added all information.  Summary, Requirements, Constraints, Domain Model, Evaluation and Recommendations. |
| 1.1 | 11/20/2024 | Janai Williams | Added revisions to Evaluation and Summary sections. Added System Architecture View. |
| 1.2 | 12/9/2024 | Janai Williams | Made slight revisions to the Recommendations 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 own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to expand Draw It or Lose It, originally an Android game, into a web version that works on various devices (computers, tablets, and phones). The goal is to keep the game simple to use, make sure it performs well on all platforms, and keep users' data secure. This proposal outlines our plan to meet these needs.

## Requirements

The game needs to:

* Allow multiple teams with multiple players.
* Ensure that each game, team, and player has a unique name.
* Only allow one instance of the game in memory at a time, to prevent duplicate sessions.
* Be accessible from different types of devices and operating systems (Windows, Mac, Linux, mobile).

## [Design Constraints](#_2et92p0)

Building this web game comes with some challenges:

* Compatibility: The game has to work well across different devices and operating systems.
* Unique Game Instance: Only one game session should be active at a time, so we need careful controls to avoid duplicates.
* Unique Naming: Each game, team, and player needs a unique name to prevent conflicts. These challenges mean we need to be careful in our design to ensure the game runs smoothly for everyone, no matter the platform, and doesn’t create duplicate sessions.

## [System Architecture View](#_ilbxbyevv6b6)

The game will have three main parts:

* Server: Manages the game logic and stores data like teams, players, and scores.

Runs on a Linux server for stability and cost-effectiveness.

* Client: This is what players see and use to play the game.

Built as a website that works on computers, tablets, and phones.

* Communication: Connects the website (client) to the server securely so data can be sent back and forth.

## [Domain Model](#_8h2ehzxfam4o)

The game’s structure is organized using classes like **Game**, **Team**, and **Player**. These classes share common features from a base **Entity** class, allowing them to work together easily. A **GameService** class acts as the central controller, handling the setup and organization of games, teams, and players. **GameService** uses the Singleton pattern, which means only one instance of this class exists, helping keep everything consistent across the application. Each **Game** can have several teams, and each team can have multiple players, creating a clear and organized structure. Helper classes like **ProgramDriver** and **SingletonTester** support and test this setup, making sure it all works smoothly. This design makes the game easy to develop and expand.

**"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** | User-friendly but costly. | Stable, free, and ideal for hosting. | Good integration but requires licensing. | Not suitable for hosting. |
| **Client Side** | Requires specialized tools like Xcode. | Cost-effective, but limited tools. | Cost-effective, but limited tools. | Requires optimization for performance. |
| **Development Tools** | Xcode and web-compatible tools (JavaScript). | Open-source tools like VS Code. | Visual Studio, React, JavaScript. | Android Studio, Xcode (iOS/Android). |

## 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**: Use Linux for hosting the game because it is cost-efficient, stable, and scalable. It is widely used in web hosting and can handle high traffic with fewer resources compared to other platforms.
2. **Operating Systems Architectures**: Linux is lightweight and efficient for hosting web applications.
3. **Storage Management**: Use cloud storage to handle game data so it’s easy to scale as the game grows. Cloud storage makes it easier to scale as the game grows and ensures data backups for security and disaster recovery.
4. **Memory Management**: Linux manages memory well, so it can efficiently handle a single instance of the game, preventing memory overload.
5. **Distributed Systems and Networks**: Use a cloud-based hosting provider to enable global access and ensure low latency for all users. Using dedicated servers or cloud services will help keep things fast and reliable.
6. **Security**: Implement HTTPS and secure authentication to protect user information. Regular updates and monitoring will help ensure ongoing security.