

Draw It or Lose It

# **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 | 07/20/25 | Josef Zoucha | Provided software design for Draw It or Lose It |

**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* would like to expand their Android game, ‘Draw It or Lose It’, into a web-based platform that supports many environments. The platform will need to be scalable and secure to handle the user traffic they are anticipating. It will also need to manage unique instances of game names and team names and players whole also ensuring efficient memory usage. In using object-oriented principles and software design patterns, like the Singleton pattern for our game management, we can ensure a single game instance in memory.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check if a name is already in use.
* Only one instance of the game can exist in memory at one time.

## [Design Constraints](#_2et92p0)

* Single Instance Memory: Only one instance of each game can exist in memory at one time. By using the Singleton design pattern, we can control the game data.
* Platform Independence: The game needs to be functional across multiple operating systems and web browsers.
* Unique Identifiers: Each game and team name must be unique to avoid memory and data issues. We will need a system to validate names for this.
* Network Reliability: Communication between client and server must be optimized to minimize connection issues.
* Security: Secure data transmission across numerous platforms is needed.
* Scale: The system must be able to keep up with multiple users and teams being made simultaneously.

## [System Architecture View](#_ilbxbyevv6b6)

* Client Tier: A browser-based UI for gameplay.
* Application Tier: A server-side layer ensuring uniqueness and managing the game data using singleton rules.
* Data Tier: In-memory structures for storing game data.

## [Domain Model](#_8h2ehzxfam4o)

* Player: Represents the individual player in the game.
* Team: Holds ‘Player’ objects which allows multiple players to be on a team.
* Game: Holds ‘Team’ objects and allows adding teams.
* GameService: This class implements the singleton pattern which ensures only one instance manages games by managing all the counters and names for teams and players.
* Entity: Base class that promotes inheritance and reusability. Other classes extend this one.

The relationship between classes in this program shows composition and allows building complex structures off simple ones. The object-oriented principles being used provide uniqueness to names and provides a centralized management.

**"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** | macOS can host web applications, but it's less commonly used in production environments. It's better suited for local development or staging environments. Licensing costs are higher, and server support is limited. | Linux is the most popular platform for web servers (e.g., Apache, NGINX). It’s open source, highly scalable, secure, and supported by major cloud providers. No licensing costs. | Windows Server supports ASP.NET and IIS. It's enterprise-friendly and integrates well with Microsoft tools. However, it comes with higher licensing and maintenance costs. | Mobile devices aren’t used for production hosting due to limited resources. They can be used for lightweight testing, but not suitable for scalable web hosting. |
| **Client Side** | Great for frontend development and testing across Safari and Chrome. macOS also allows building for iOS, making it essential for mobile cross-platform testing. | Ideal for backend/server development. Browser testing is more limited but possible using headless browsers or tools like Selenium. | Most widely used desktop OS, supports full browser and application testing. Compatible with many tools and widely supported. | Essential for testing mobile responsiveness. Requires testing across both Android and iOS platforms using emulators, simulators, or real devices. Platform fragmentation must be considered |
| **Development Tools** | * Java * JavaScript * HTML/CSS * InteliJ IDEA * Xcode * Eclipse * Docker * Git | * Java * JavaScript * Spring Boot * Eclipse * NetBeans * HTML/CSS * Spring Boot * Git | * Java * JavaScript * InteliJ * Eclipse * Visual Studio * Git * HTML/CSS | * Android Studio * Swift * Flutter * Xcode * JavaScript * HTML5 |

## 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**: Linux is my recommendation to use on the server side as an operating platform for Draw It or Lose It. It is scalable and cost efficient and provides strong support for web-based applications.
2. **Operating Systems Architectures**: Linux uses monolithic kernel architecture that has great performance and memory management.
3. **Storage Management**: We can use MySQL to integrate a relational database for storage of returning players, team and game names. We can also use resources like the cloud to save the save information
4. **Memory Management**: Linux uses padding, demand loading and virtual memory, and using the singleton pattern ensures there is only one instance of a game which conserves memory.
5. **Distributed Systems and Networks**: We can use RESTful APIs for client server communication. Using tech like WebSockets can give real-time feedback about gameplay, and load balancing should be kept in mind to handle scale and outages.
6. **Security**: HTTPS encryption for secure data transmission should be used. Input validation and user authentication should be implemented to protect user information. Linux has firewalls and access control lists which we can use for security purposes.