

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 | 05/25/25 | Mike Garcia | Initial draft of software design document |

**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 have tasked Creative Technology Solutions (CTS) to develop a web-based game that serves multiple platforms based on their current Android only game, Draw it or Lose It. This document outlines a software design that will incorporate object-oriented principles and design patterns. The design pattern includes a Singleton pattern which will ensure scalability, maintainability, and performance. The design will use a GameService class in order to manage unique instances of games, teams, and players. Each entity will extend from common Entity for consistency. This approach will provide a strong foundation for future enhancements.

## Requirements

*Business requirements:*

* *Provide a multiplatform, web-based version of Draw It or Lose It*
* *Game should allow multiple teams and players per team*
* *Game should have unique naming for games and teams*
* *Only one instance of a game should be in memory at a time*

*Technical Requirements:*

* *Web-based implementation using scalable architecture*
* *Use of unique identifiers for games, teams and players*
* *Support for real-time gameplay*
* *Ensure secure data handling and user session management*

## [Design Constraints](#_2et92p0)

**<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>**

1. Single Instance Control: Only one instance of game should exist at a time. This is done using a Singleton pattern in the GameService class.
2. Unique Identifiers: Each name for games and team must be unique. The system verifies this through the UI and service levels
3. Cross-Platform Compatibility: The game must be able to work across multiple platforms such as desktop and mobile browsers. This will require the front end to use responsive design and technologies.
4. Network Latency and Reliability: Because the game is a real-time game, the system must be able to manage an asynchronous communication with minimal lag.
5. Scalability: The architecture must accommodate growth in users and games without decreasing the performance

## [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)

**<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>**

**"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.**

* Entity is an abstract base class that contains fields like id and name which are inherited by *Game*, *Team*, and *Player*
* GameService is a Singleton class, indicated by the getInstance() method the private stative service attribute
* *Game* contains a list of *Team* objects, and *Team* contains a list of *Player* objects which demonstrate composition.
* The use of getters, constructors and toString() methods throughout supports encapsulation, hiding internal data while providing controlled access.

## [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 host server-side applications with Apache or Node.js. However, there is limited support due to higher hardware cost | Preferred choice for server environments due to its open-source nature. | Supports IIS and other tools but is often considered less secure. It is also more resource heavy than Linux | Mobile devices are not ideal for hosting. They are usually more suited as clients due to limited processing power, battery life and network reliability |
| **Client Side** | Must consider Safari compatibility. Uses exclusive Development tools which add complexity | May require additional testing dues to varied browser and desktop environment behavior | Supports a wide range of browsers and environments. Ideal for testing compatibility. Tools are widely available | Requires responsive UI design and may involve using native features. Testing across devices may increase time and cost |
| **Development Tools** | Languages:  JavaScript, TypeScript  IDEs:  Visual Studio Code, IntelliJ IDEA  Tools:  Homebrew, Docker | Languages:  JavaScript, Java Python  IDEs:  VS Code, Eclipse  Tools:  Git, Docker, npm | Languages:  C#, JavaScript, Python  IDEs:  Visual Studio, VS Code  Tools: IIS, Docker, git | Languages:  Kotlin (Android), Swift (iOS), Flutter (Cross-Platform)  Tools: Android Studio, Xcode, React Native |

## 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 recommended as the server platform for its cost efficiency, performance and support for web hosting technologies
2. **Operating Systems Architectures**: Three-tier Architecture: Presentation (Web interface), application (logic in GameService), and data (database for persistence).
3. **Storage Management**: Cloud-based relational database.
4. **Memory Management**: The Application server should employ dynamic memory allocation to manage game instances. Garbage collection will also be used to clean up any unused objects.
5. **Distributed Systems and Networks**: WebSocket or Socket.IO are recommended in order to support real-time updates. The app should be hosted using cloud platforms to ensure high availability and load balancing.
6. **Security**: you can use HTTPS for all data transmission for secure communication between clients and the server. User session can be handled using JWT tokens, and encryption. The server will also use firewalls and access control mechanisms in order to protect the system from unauthorized access.