

<Name-of-Software-Application>

# **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 | 05/22/25 | Adil Patel | Initial submission with complete software design document. |
| 2.0 | 06/12/25 | Adil Patel | Updated Evaluation and Recommendations for Project Two submission. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has requested the development of a web-based version of its game, Draw It or Lose It, currently available only on Android. The main goal is to enable cross-platform accessibility while ensuring performance, security, and a smooth user experience. This document proposes a Java-based software design using design patterns such as Singleton and Iterator to manage instance control and ensure uniqueness of game elements. The approach is tailored to be scalable, maintainable, and aligned with web-based distributed systems.

## Requirements

• Allow multiple teams per game  
• Each team has multiple players  
• Unique names for game, team, and player  
• Only one game instance in memory   
• Ability to check if names are already in use

## [Design Constraints](#_2et92p0)

The application must function in a web-based distributed environment, which requires ensuring that only one instance of the game runs in memory using the Singleton pattern. It also demands maintaining unique naming across entities and robust communication between multiple clients, possibly across platforms. The system must handle concurrent sessions, high availability, and consistent state synchronization across distributed components.

## [System Architecture View](#_ilbxbyevv6b6)

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 shows an Entity superclass with Game, Team, and Player as subclasses. Each class inherits the common attributes 'id' and 'name'. The Game class contains a list of teams, and the Team class contains a list of players. Object-oriented principles include inheritance encapsulation and design patterns. This structure supports scalability, code reuse, and maintainability while meeting the software requirements.

**"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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac systems can host web applications effectively using tools like Apache or Nginx. However, they are less common in enterprise server environments due to licensing costs and hardware limitations. | Linux is the preferred server OS for web-based applications due to its open-source nature, high stability, security, and performance. It supports a wide range of web servers and is cost-effective for scalable deployments. | Windows servers are widely used in enterprise settings, particularly for .NET applications. They offer GUI and integration with Microsoft tools but have higher licensing costs and slightly less flexibility than Linux. | Mobile devices are not suitable for hosting web-based applications due to limited processing power, battery life, and lack of server software support. |
| **Client Side** | Mac clients require ensuring compatibility with Safari and possibly Chrome. Development may require Xcode for macOS,specific apps. Mac users typically represent a premium user base, so testing on Mac is important despite more development costs. | Client-side Linux support is uncommon for general consumer software but is important for technical and developer audiences. Supporting Linux desktops may require additional testing for browsers like Firefox. | Windows is the most common desktop OS, so it must be fully supported. Development and testing on Windows is critical and cost-effective using widely available tools like Visual Studio and Eclipse. | Support for Android and iOS is essential due to mobile market dominance. Development requires platform-specific tools Android Studio, Xcode . |
| **Development Tools** | Common tools include IntelliJ IDEA, Eclipse, and Xcode. Java is supported natively. Xcode is used for building macOS and iOS clients. Git and Terminal are also useful for managing code. | Supports development with tools like Eclipse, VS Code, and command-line. Excellent for Java development and scripting. | Development tools include IntelliJ, Eclipse, Visual Studio, and NetBeans. Java, C#, and .NET are all supported. Suitable for both desktop and web development with strong IDE support. | Mobile devices are not development platforms but are tested using emulators or physical device debugging. Android Studio and Xcode are used on desktops to create mobile apps for these platforms. |

## 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**: I recommend using Linux as the primary operating platform for expanding Draw It or Lose It. Linux provides a scalable, secure, and cost-effective foundation for hosting web-based applications. It supports a wide range of development tools, has excellent community and enterprise support, and is the industry standard for high-availability systems.
2. **Operating Systems Architectures**: The recommended Linux-based architecture follows a modular approach using a client-server model. Server-side applications will run on a Linux server with Apache for HTTP handling. Client-side applications will communicate via REST APIs or WebSocket protocols, with potential use of containers to isolate services.
3. **Storage Management**: A cloud-based storage system like Amazon or Azure Storage can be integrated with the Linux platform to store game assets, drawings, and user data.
4. **Memory Management**: Linux uses virtual memory management with demand paging, allowing it to manage physical and virtual memory efficiently. The operating system leverages swap space and the page cache to optimize performance.
5. **Distributed Systems and Networks**: Draw It or Lose It can use RESTful APIs or WebSockets for real-time communication between clients on different platforms.
6. **Security**: User data should be encrypted in transit and at rest using platform level encryption Linux supports secure user authentication.