

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 | 12/14/24 | Johanna Rodriguez | Initial draft of the 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 needs a software solution to manage and scale the game Draw It or Lose It across different platforms. The goal is to create a distributed application that can handle games, teams, and players. This solution will utilize object-oriented principles and a singleton pattern to ensure efficient and centralized control of game data, while ensuring scalability and adaptability. The software will be designed to run smoothly across different platforms, including Windows, Linux, Mac, and mobile devices, while maintaining security and ease of use.

## Requirements

**Business Requirements:**

* The software must support managing many games, teams, and players.
* The application must ensure that each game, team, and player have a unique name.
* The software should work seamlessly across different platforms, including desktop and mobile devices.

**Technical Requirements:**

* The application will use object-oriented design principles for efficient development and future scalability.
* A **singleton pattern** will be used to manage the game data centrally.
* The structure of the software should allow easy updates and future feature expansions.

## [Design Constraints](#_2et92p0)

**Distributed Environment:**

* The application will operate in a distributed, web-based environment, which will require efficient management of concurrent requests and data consistency.
* Latency, bandwidth issues, and security concerns will need to be addressed to ensure smooth operation across different platforms.

**ID Generation:**

* The application must generate unique IDs for games, teams, and players. These IDs must be thread-safe and globally unique to avoid conflicts in a distributed environment.

**Cross-Platform Compatibility:**

* The software must be compatible with multiple operating systems (Windows, Mac, Linux) and mobile devices, which will require platform-specific optimization and extensive testing.

**Security Requirements:**

* All communication between the client and server must be encrypted to prevent unauthorized access.
* User data, such as player names and scores, should be securely stored and protected.

## [System Architecture View](#_ilbxbyevv6b6)

Since the requirements state a distributed system, the application will likely use a client-serverarchitecture. The server will host the game data and provide services to the client-side applications, which will run on users’ devices. Communication between the server and clients will be handled through APIs or WebSocket’s to ensure real-time interaction. Data storage will be handled by a relational database such as MySQL or PostgreSQL to ensure consistent and scalable storage of game-related information.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram for the system shows the relationships between several key entities: Game, Team, and Player. These entities inherit from a base class Entity, which holds common properties like id and name. The GameService class is designed as a singleton, ensuring that only one instance manages the game-related data and operations.

**Object-Oriented Principles:**

**Inheritance**: The Entity class allows for shared attributes and methods, reducing redundancy.

**Encapsulation**: Private fields ensure controlled access to data, allowing the internal workings of the classes to remain hidden from other parts of the system.

**Polymorphism**: The use of methods like addGame(), addTeam(), and addPlayer() shows how the same interface can be used for different entity types while adapting to the specific entity needs.

**Singleton Pattern**: The GameService class ensures that only one instance manages the entire game system, which simplifies managing game data across multiple clients.

**"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** | Mac can handle server hosting reasonably well, but its hardware and licensing can be expensive. | Linux is the go-to choice for servers. It’s reliable, free, and works well for hosting web-based applications. | Windows Server is user-friendly and works well for companies using Microsoft tools, but the licensing cost can be a downside. | Mobile devices are better suited for running apps rather than hosting servers. |
| **Client Side** | Developing for Mac is relatively easy if familiar with Apple tools, but it requires testing on Safari to ensure compatibility. | Linux is flexible but may require more effort to ensure everything looks and works well, especially for Firefox users. | Windows offers many development tools, but attention to updates and compatibility with browsers like Edge is important. | Mobile platforms require lightweight apps that must be optimized for both iOS and Android. |
| **Development Tools** | Xcode or IntelliJ IDEA are popular development tools on Mac, but some advanced features may require a paid license. | Linux offers free tools like Eclipse or VS Code, but they may require more time to learn | Windows has powerful tools like Visual Studio, but some may require a paid license. | Mobile apps are typically developed with Android Studio or Xcode, and tools like Flutter simplify development for both 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**:

We recommend using **Linux** as the operating platform for hosting the game. It is free, stable, and widely used for web-based applications, which makes it a cost-effective and reliable choice for **Draw It or Lose It.**

1. **Operating Systems Architectures**:

The software should use client**-server architecture**. The **Linux server** will handle all the game data, and clients on various platforms (Windows, Mac, Linux, and mobile devices) will connect to the server to interact with the game. Communication will be done through **RESTful APIs** or **WebSocket’s** for real-time updates

1. **Storage Management:**

For scalable and consistent data storage, a **relational database** such as **MySQL** or **PostgreSQL** should be used. These databases will ensure that data is properly managed and that **unique IDs** for games, teams, and players are maintained.

1. **Memory Management**:

Linux provides robust memory management features that ensure the application runs smoothly even with multiple players accessing the game simultaneously.

1. **Distributed Systems and Networks**:

To handle communication between platforms, we will use **APIs** and **WebSocket’s** for real-time interaction. Additionally, **load balancers** will be used to distribute requests evenly across the server, maintaining up time and availability during heavy traffic.

1. **Security**:

* TLS/SSL encryption will be used to secure data in transit between clients and the server.
* Sensitive information such as passwords will be hashed and stored securely.
* Regular security audits will be performed to identify and fix any potential vulnerabilities.