

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 | 10/21/25 | James Piggot | Initial Version of the software Design Document 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)

This document presents the software design for the gaming Room’s web-based game, “Draw it or lose it” The solution proposes a backend management for game logic utilizing object-oriented principles. The design relies on a Singleton pattern to ensure only a single instance exists in memory. The iterator will be used to enforce unique games and team names. The Final design establishing a foundation for application’s core logic.

## Requirements

Business Requirements:

* Develop a web-based game accessible on multiple platforms.
* The game, "Draw It or Lose It," will feature multiple teams guessing drawings.
* Each game will consist of four one-minute rounds.
* Drawings will be rendered gradually and be fully complete at the 30-second mark.
* A bonus round allows remaining teams one guess with a 15-second time limit.

Technical Requirements:

* The application must support games with one or more teams.
* Each team must be able to have multiple players.
* All game and team names must be unique.
* Only one instance of the game can exist in memory at any given time.
* Unique identifiers must be created for each game, team, and player.

## [Design Constraints](#_2et92p0)

* Distributed Environment: The web-ased nature of the games means there will be various connections made from many locations. This will introduce lag, latency, network reliability and synchronization among the players.
* Concurrency: The system will need to be able to handle more than one user interacting with the game at the same time, including conflicting data to prevent corruption.
* Scalability: We will need to concern ourselves with growing popularity of the game to make sure it can support a growing nature of players and users. It should be easily adaptable.
* Security: Web based applications are always exposed to security threats, while this is only a game, we must be aware of malicious actions and attempts to maintain the game integrity and prevent cheating.
* Performance: The game’s real time nature along with the involvement of latency will require high performance and everything must be high quality and introduce a client without lag.

## [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)

The UML Diagram bellow represents the core model for the “Draw it or lose it” game application. The six main classes being, Entity, Team, Player, Game, Game service, program driver.   
  
**Class Relationships:**  
The classes exist mainly as an inheritance relationship where Game, Team and Player will all inherit from the base class of Entity. This is indicated by the solid line with a arrowhead. This means that entity will serve as the basic blueprint for the other related classes.  
There is a series of several associations between the remaining classes of (0..\*) this serve to show that one class contains a collect of other object. For example, GameService is associated with Game, and it can contain zero or more game objects. Game is associated with team and can contain zero or more teams. And last team is associated with player and can have zero or more players.  
**Object-Oriented Principles in use:**

**Inheritance:** The entity base class serves to set the basic logic and attributes for the other classes instead of every class needing to declare them.

**Encapsulation:** each class encapsulates its data and behavior and can only be accessed their public methods to protect unauthorized modification.

**Singleton Pattern**: Gameservice is a singleton means that only one instance of Gameservice can be ran at once for any given memory.  
**Iterator Patterns:** The methods such as addGame(), getGame(), addTeam() and addPlayer) use an iterator pattern to go through their collection of objects, by iterating through the existing game and team names before adding new instances in a unnecessary way.

**"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** | Macs are great for development, but they're not a good choice for a production server. They're expensive and not widely used for this kind of work, which means less community support. | Linux is our top recommendation. It's the industry standard for web servers, and it's free, super stable, and can handle a huge number of users. You just need some expertise with the command line. | Windows is a decent option if you're already familiar with it. The graphical interface is user-friendly, but it comes with licensing costs and isn't as efficient or scalable as Linux for web hosting. | Mobile devices simply should not be used as servers. They do not have the power, networking capabilities or many other things to run the game at a high-performance. |
| **Client Side** | Our game is web-based, so for Mac, Linux, and Windows, the development approach is the same: we build for web browsers. This means we can use the same code and tools across all three platforms, which saves time and money. | Our game is web-based, so for Mac, Linux, and Windows, the development approach is the same: we build for web browsers. This means we can use the same code and tools across all three platforms, which saves time and money. | Our game is web-based, so for Mac, Linux, and Windows, the development approach is the same: we build for web browsers. This means we can use the same code and tools across all three platforms, which saves time and money. | For Mobile Devices, we have to design with smaller screens in mind. This means the layout needs to adapt automatically, which takes more time. The expertise needed is in responsive and mobile-first design, but we can still use our core web technologies to build it. |
| **Development Tools** | We'll use Java for the backend and standard web languages (JS, HTML, CSS) for the frontend. Our go-to tools will be IDEs like IntelliJ IDEA or VS Code, along with Git for version control. | The core languages are the same, but we'll rely on the command line and Linux's package manager to set up the Java Development Kit (JDK) and other necessary software. | Windows is straightforward since our tools like Java and IntelliJ IDEA work across platforms. We can also use WSL (Windows Subsystem for Linux) to help with server-side work. | We'll build the mobile interface using web technologies (JS, HTML, CSS). For testing, we'll use emulators and tools like Android Studio to make sure everything runs smoothly on phones and tablets. |

## 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**: Due to the nature of this game being web-based, Linux is the most suitable platform for this project. It’s highly regarded for web servers and is open-sourced and highly scalable for future upgrades. This will become critical as the game is multi-user, it is also the most cost-effective option in comparison to windows due to no licensing fees.
2. **Operating Systems Architectures**: Due to the nature of this being a web-based game, the only time we need to worry about this is within mobile devices. So, we can focus our options on things that are all cross platform to ensure that we can handle each option.
3. **Storage Management**: For storage management I recommend a NoSQL database, these are commonly used and are well suited for our Linux environment choice. Making a great fit for face-paced real time game. It is also very scaleable for the future.
4. **Memory Management**: In a Linux environment, memory management is a complex but highly optimized process handled by the kernel. The kernel's memory manager is responsible for allocating and deallocating memory, managing virtual memory, and ensuring that each process has the memory it needs without interfering with others. The operating system uses virtual memory, providing each process with its own private address space, which is then mapped to physical RAM. If physical memory becomes scarce, less-used portions are moved to swap space on the hard disk. Our Java backend will run on the Java Virtual Machine (JVM), which handles its own internal memory allocation and garbage collection, automatically reclaiming unused memory to prevent leaks. The Linux kernel and the JVM work together to ensure efficient and reliable memory usage for our real-time game application.
5. **Distributed Systems and Networks**: The "Draw It or Lose It" game is a distributed system, with multiple client devices communicating with a central server over a network. This client-server architecture will use WebSockets for real-time drawing updates to ensure low-latency gameplay, while standard HTTP/S will handle other data transfers. The primary dependencies are a reliable network connection and the central server, which acts as the single authoritative source for all game state. While the client will attempt to gracefully handle temporary network outages, a server-side outage would cause all active game sessions to be lost, as the single-instance GameService is a key dependency and a single point of failure in the current design.
6. **Security**: Protecting user information is paramount in this web-based distributed system, which will be built on the robust Linux operating platform. All communication between the client and server will be encrypted using HTTPS and WSS (WebSocket Secure) to protect data in transit from eavesdropping. On the server side, the NoSQL database will use access controls and encryption to secure data at rest. We will further harden the system with a strict firewall, run the application process under a non-root user for least privilege, and implement server-side input validation to prevent injection attacks and ensure the integrity of user data.