

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 4/20/2025 | Mathew Masar | Final iteration for customer |

**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)

Creative Technology Solutions has been contracted by The Gaming Room to modernize and expand their Android based game Draw It or Lose It into a web-based, cross-platform application. The game will involve four timed rounds where a team will guess what image is being produced from stock images, with images rendering steadily and partially over a 30 second period. After the 30 second period, other teams will have 15 seconds to guess the image if the original team did not do so in the 30 second period. The objective of Creative Technology Solutions is to design a flexible and scalable application that supports multiple games, teams, and players with unique identifiers to avoid name conflicts.

In order to ensure these goals are met, Java will be used to create this object-oriented application. Key elements to be included will be Inheritance for shared entities, a Singleton pattern to ensure one instance of the game management service, and the Iterator pattern to ensure unique identifiers for games, teams, and players.

## Requirements

* A game will have the ability to include one or more teams.
* Each team will support multiple players.
* All game, team, and player names must be unique within their scope.
* Only one instance of the game management service can exist (singleton).
* Each entity (game, team, player) will have a unique identifier assigned automatically.
* The application must be scalable to support multiple users and future enhancements.
* Object-oriented principles must be used to support maintainability.

## [Design Constraints](#_2et92p0)

**Web based distributed environment:**

* Simultaneous access from different clients
* Synchronized game state handling
* Scalable architecture for concurrent users

**Singleton pattern for GameService:**

* Ensures a single and consistent instance of the game manager
* Prevents conflicts or inconsistent data between game sessions

**Unique identifiers and names:**

* Enforced using iterators during object creation (validation)
* Prevents duplication and confusion across the system

**Immutability for Entities:**

* Names and IDs are final after creation
* Simplifies debugging and improves consistency

**Object oriented principles:**

* Inheritance, encapsulation, and modular design followed
* Enhances maintainability and extensibility of code

## [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 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 (Base Class):**

* + Contains shared attributes id and name

**Game (extends Entity):**

* + Holds a list of Team objects
  + Adds new teams only if their names are unique

**Team (extends Entity):**

* + Holds a list of Player objects
  + Adds new players only if their names are unique

**Player (extends Entity):**

* + Represents individuals on a team

**GameService:**

* + Implements Singleton pattern
  + Manages games, teams, and players

**ProgramDriver & SingletonTester:**

* + Program entry point and singleton verification utility tool

**Object Oriented Principles Used:**

* Inheritance for common behavior and reduced duplication
* Encapsulation of internal data structures
* Iterator pattern to validate unique names

## [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** | Offers robust support via Apache, Docker, and NGINX.  Offers POSIX compliance and Unix based stability. Good for local dev environments and testing but can be expensive and restrictive for enterprise scale production web hosting. | Open source, strong security, lightweight, and widely used for hosting Java web apps. Supports threading with Pthreads and has efficient CPU scheduling, making it highly scalable and reliable for web apps. Highly preferred in production. | Widely used in enterprise with good support and integration for ASP.NET and Java but is more resource heavy. Uses 1:1 threading model and has advanced GUI based tools for deployment. | Android and iOS are not suited for hosting but good for testing. Emulators and device testing tools are effective for testing and validating client side interactions with the server. Extremely lightweight and local servers can be used for development purposes. |
| **Client Side** | UI/UX dev is intuitive with Xcode and provides solid browser support, but limits flexibility. Super strict app store policies and variations in system libraries require very thorough testing. | Requires additional libraries for front end development. Not ideal for UI heavy applications but supports most modern browsers and standards. Favored by developers but could be challenging to use for non-technical users in desktop environments. | Excellent support for front end web development with support for HTML/CSS/Javascript and modern browsers. Tools like Visual Studio help streamline development. Excellent multithreading enhances responsiveness for browser based games. | Must consider screen size and OS variability. Different scheduling models are used by different mobile OSs and may disturb or kill background threads. Requires responsive design and cross platform frameworks like Flutter or React Native to ensure consistent user experience. |
| **Development Tools** | IntelliJ, Eclipse, Xcode for GUI, Gradle/Maven. Java dev is seamless per UNIX foundations. Tools available for threading and CPU scheduling. Back end Java based deployment can be more complex than on Linux. | Same as Mac, plus NetBeans, plus native bash tools and lightweight IDEs like Geany. Strong support for Pthreads and open source libraries. Lightweight IDEs and command line tools improve efficiency and minimal licensing costs make it very cost effective. | IntelliJ, Eclipse, Visual Studio, NetBeans; best GUI IDE support. Excellent debugging, threading visualization, and performance analyses tools. Some tools require licensing costs, the OS itself may also have higher costs. | Android Studio, VS Code, Xcode. Must choose cross platform frameworks for unified user experience. Developers must be aware of differing thread and memory models between Android and iOS and thorough testing must be done to optimize and standardize performance and security across 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**:

Linux would be the preferred operating platform for hosting Draw It or Lose It based on performance, scalability, and compatibility & strong support with Java based web applications. Linux being largely open source in nature will help provide cost efficiency as developers will have a strong starting point to begin with. Characteristics like POSIX compliance and robust support for threading, process scheduling, and resource & memory management make it suitable for distributed systems. Seamless integration with modern cloud platforms like AWS, Azure, and Google Cloud also make Linux widely used in production environments. Altogether, a Linux hosting platform will allow the requirements for simultaneous user access, low-latency performance, and reliable backend processing.

1. **Operating Systems Architectures**:

Linux supports a hybrid operating system architecture that blends characteristics of both monolithic and microkernel designs. A monolithic architecture means that core functions like memory handling, file systems, and process management all run together in a single space, allowing for fast and efficient execution. On the other hand a microkernel separates these services into smaller, independent components to improve modularity and fault isolation. Linux uses a monolithic kernel at its core but maintains flexibility through loadable kernel modules, which gives it some of the advantages found in microkernel systems without sacrificing performance.

This architecture would work for Draw It or Lose It as it supports multitasking, lightweight threads, and efficient resource sharing across user sessions. POSIX compliance also ensures compatibility across other Unix systems which is helpful for development and deployment across varying platforms. These architectural characteristics allow Linux to handle high volumes of concurrent users, support real time gameplay, and maintain system stability in a distributed environment.

1. **Storage Management**:

Draw It or Lose It relies on both large static image assets and dynamic game related data. Having a clear separation between types of storage is important. Static image files such as the ones used during each round are best stored using a cloud based object storage service like AWS S3, Google Cloud Storage, or Azure Blob Storage. These platforms are built to handle large volumes of unstructured data and provide fast, scalable access from anywhere.

In contrast, structured data like game info, user profiles, team details, and overall game progress should be handled by a relational database system such as PostgreSQL or MySQL. Systems like these are ideal for managing organized data, maintaining relationships between records, and ensuring data consistency across the application. Separating storage in this way helps optimize performance and scalability, especially as the game expands to support more users and active sessions. Additional techniques like metadata caching, file prefetching, and directory indexing can also help improve access speed and reduce latency during gameplay.

1. **Memory Management**:

Memory management plays a crucial role in ensuring Draw It or Lose It performs smoothly, especially when handling high resolution images and multiple users at the same time. On a Linux based platform memory is managed through virtual memory techniques, paging, and caching. These features allow the system to efficiently allocate and free memory as needed, helping to reduce lag and avoid crashes during gameplay.

The JVM adds another layer of memory handling. It uses garbage collection to automatically manage heap memory and remove unused objects from memory over time. This helps reduce memory leaks and improves overall system performance during long-running sessions. When combined with image preloading and caching frequently used assets in memory, these strategies work together to reduce latency and provide a better experience for users.

Linux also supports lightweight threading through POSIX Pthreads, allowing for efficient multithreading and resource sharing across multiple users and / or sessions. This is especially helpful in maintaining game state updates and rendering tasks in real time without putting too much strain on the system. When working with shared resources proper synchronization is important to prevent data races where multiple threads try to read and write to the same memory at the same time without coordination. Avoiding data races will help keep game sessions stable and predictable, especially under high user activity.

1. **Distributed Systems and Networks**:

The application should use RESTful APIs for communication between clients and the server. JSON payloads and HTTPS protocols ensure platform-agnostic interaction (services that function seamlessly across different platforms). Load balancers and failover configurations help maintain uptime and performance.

Since Draw It or Lose It needs to run across multiple platforms and allow users to interact in real time, distributed systems and network communication are crucial to making everything work together seamlessly. A RESTful API is recommended as the main method of communication between client devices and the server. REST is lightweight, platform independent, and works well with JSON data, making it a good fit for games that need to handle a lot of back and forth between users and the server.

To maintain uptime and responsiveness, the system should use load balancers and failover configurations to keep traffic flowing even during high demand or unexpected downtime. These features help distribute user requests evenly and allow another instance to take over if one server becomes unresponsive.

Maintaining consistent game state across devices also means that dependencies like user sessions, image progress, and timers need to stay synced. Techniques like token based authentication and session tracking can help manage that. Token based authentication is helpful in this setup because it allows each request to be verified without storing session data directly on the server. Tokens are lightweight, secure, and can include encoded information like user identity or session validity. This makes them ideal for distributed environments where users might move between different devices or server instances. Overall, using a distributed system with strong network support will give users a reliable experience no matter where they connect from or what platform they’re using to play the game.

1. **Security**:

Security is a must have for Draw It or Lose It especially with users interacting across multiple platforms and sharing personal information. All communication between users and the server should take place over HTTPS to keep data safe during transmission. Encrypting things like user IDs and session tokens adds another layer of protection in case any data is exposed or intercepted.

Input validation is also important to help prevent common attacks like SQL injection or cross site scripting, both of which could put the system or user data at risk. Setting up role-based access control makes sure that only the right users can perform sensitive actions like managing game sessions or modifying player data.

Running regular vulnerability scans and keeping the system up to date will help patch any known security issues before they can be exploited. With these practices in place the game will be in a strong position to protect user information and deliver a safe experience across all platforms.