

Draw It or Lose It

# **CS 230 Project Software Design**

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 | 06/19/2025 | Christopher Bryan | 3rd Draft |

## [Executive Summary](#_sbfa50wo7nsh)

*The Gaming Room is expanding its Android-exclusive game, Draw It or Lose It, into a web-based, cross-platform application. The game is a team-based drawing and guessing competition played in timed rounds, where players try to identify puzzles from gradually rendered images.*

*To support this transition, a software design leveraging object-oriented principles (OOP) and the singleton pattern is recommended. This architecture will enable centralized game management, ensure unique identification of games, teams, and players, and enforce memory efficiency by limiting service instantiation. These choices provide a scalable, maintainable solution suited for cross-platform deployment and future feature enhancements.*

## Requirements

***Business Requirements:***

* *Expand the existing game, Draw It or Lose It, beyond its current Android-only implementation to a fully cross-platform application accessible on multiple devices and operating systems. This includes developing responsive web versions compatible with Windows, macOS, and Linux desktops*
* *Preserve core gameplay experience across platforms by ensuring consistent image rendering speed, intuitive user interface interactions, and accurate timing during each round of play. For example, images should be rendered progressively and be fully visible at the 30-second mark regardless of device type*
* *Ensure scalability so that future development can include features such as real-time multiplayer functionality, a global leaderboard, and cross-platform synchronization of player data. Additionally, scalability is required to accommodate a growing user base without performance degradation*

***Technical Requirements:***

* *Allow one or more teams per game with multiple players per team*
* *Make sure that game and team names are unique and can be set up easily*
* *Allow only one instance of the game service to exist in the memory at any time utilizing the singleton pattern*
* *Implement unique identifiers to be used for each game, team and player*
* *Utilize OOP principles to ensure efficient and maintainable code structure*
* *The game logic should support web deployment*
* *Reliable iteration and validation when adding new games, teams or players*

## [Design Constraints](#_2et92p0)

***Platform Distribution:*** *Draw It or Lose It must work across multiple platforms, to include web, desktop and mobile platforms. This necessitates using Java for modular layered design to enable platform independent backend logic.*

***Singleton Pattern:*** *To limit only one instance of the GameService to exist at a time, this will limit memory usage and can simplify access control.*

***Name Uniqueness:*** *Teams and games must have unique names, which will require a need for a look up mechanism and validation logic, to help with multiple user requests.*

***Scalability:*** *Initial design to support scalability for further expansion to add more teams and users, with the possibility of eventual integration into cloud-based hosting.*

***Web Compatibility:*** *Allow for communication with the web front-end and no change in user experience.*

## [Domain Model](#_8h2ehzxfam4o)

*The UML class diagram shows that the Entity class serves as a base class containing the shared attributes id and name, which are inherited by the Game, Team, and Player classes. This structure promotes code reuse and reduces redundancy. The Game class maintains a list of Team objects, enabling one or more teams to participate in a single game instance. Each Team contains a list of Player objects, allowing multiple players per team. The Player class represents individual users and inherits identity attributes from the Entity class.*

*The GameService class is responsible for managing all instances of games, teams, and players. It follows the singleton design pattern, ensuring that only one instance of GameService exists in memory at any given time. The class uses static counters to generate unique identifiers for each entity type and provides methods such as addGame() and addTeam() that implement the iterator pattern to enforce name uniqueness when creating new games or teams.*

*Encapsulation is demonstrated in GameService by declaring internal data such as games, nextGameId, nextTeamId, and instance as private, preventing direct access from outside the class. Public methods like addGame(), getGame(), and getInstance() allow controlled interaction with this data. These methods also demonstrate abstraction, as they hide the internal logic while providing a simple interface for developers.*

*The ProgramDriver and SingletonTester classes are utility components used to test and demonstrate system functionality, including the enforcement of the singleton pattern and object creation behavior.*

* *Entity is the base class for Game, Team, and Player.*
* *Game has a one-to-many relationship with Team.*
* *Team has a one-to-many relationship with Player.*
* *GameService is a singleton class that manages the creation and access of all game-related entities.*
* *ProgramDriver and SingletonTester are support classes used for validation and testing.*

**"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** | macOS is capable of server-side web hosting but has notable limitations. High cost associated with licensing. Lacks advanced server optimization tools and scalability options. Can be used effectively for smaller or specialized deployments. | Generally, Linux is the preferred server-side web platform due to its reliability, flexibility, and extensive support community. Allows for significant customization, making it suitable for varied deployment needs. Additionally, it offers robust tools for server optimization and scalability. Linux's has no licensing fees. | Good support for .NET and Java-based applications, making them versatile for enterprise-level development. Offer integration tools for Microsoft services like Azure, Office 365, and Active Directory. These can facilitate seamless transitions and management of resources. These come with notable expenses, such as licensing fees. Windows can be efficient but potentially costly option for developers seeking a dynamic environment. | Not suitable for hosting web applications due to limited processing power and network constraints. Lacks the infrastructure needed for server-side operations, such as high-speed data transfer, advanced optimization tools, and scalability options. Best for portability, accessibility, and localized or client-side applications that complement server operations. |
| **Client Side** | Investment into Apple hardware for testing, optimizing Safari, and licensing fees for macOS and iOS deployments. Time-intensive development to ensure compatibility and performance. Expertise in using coding languages such as Xcode, Swift or other languages. Knowing Apple’s documentation and deployment processes. | Linux is generally a lower cost platform to utilize due to it being open-source and its abundance of use. It can require a significant amount of time to ensure compatibility across multiple platforms depending on the different requirements and configurations needed. Needed expertise in package management, libraries, and other areas. | Can have an increased cost due to needing multiple tools and licenses to ensure ability to work across multiple browsers. Will require more development time to ensure cross browser support, and continual updates and changes as the browsers update and change. Developers will need to have a understanding of HTML, CSS and JavaScript and possibly other coding languages. | Developing and maintaining mobile apps will incur significant costs. Familiarity with frameworks, languages, and app store guidelines is crucial for efficiency and approval. Testing on various devices to ensure functionality is also essential. |
| **Development Tools** | Programming languages could include: Swift, Objective-C, JavaScript, Python and Java.  Programming tools could include: Xcode, IntelliJ IDEA, Eclipse, CocoaPods and others.  There is also the option of Command-Line Utilities like Git. | Programming languages could include: Java, Python, C++, and others.  Programming tools could include: Eclipse, IntelliJ IDEA, PyCharm, Visual Studio Code and others.  There is also the option of Command-Line Utilities like Git. | Programming languages could include: Java, Python, C++, and others.  Microsoft SQL Server, and other Windows Tools, may require licensing fees  Programming tools could include: Eclipse, IntelliJ IDEA, PyCharm, Visual Studio Code and others. | Programming languages could include for Android: Java, Python, Kotlin and others.  Programming languages could include for iOS: Swift, Objective-C, and others.  Cross-Platform tools could include: Flutter, React.  Android IDE: Android Studio  iOS IDE: Xcode |

## 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 system for hosting the server-side application of *Draw It or Lose It*. The Linux-based server platform is the top choice as it is cost-effective (no licensing fees), strong security, and scalable that can be deployed with services like AWS or Azure and can be optimized for Java-based web applications. You have the benefit of community support with any issues you may run into. It can be run consistently across multiple platforms by utilizing programs such as Docker, allowing you to utilize Windows, macOS and Linux platforms.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel architecture, which integrates core functions such as process management, memory management, file systems, and networking directly into the kernel for optimized performance. One of Linux’s strengths is its modular design, allowing individual kernel modules or system components to be loaded, replaced, or updated independently. This improves system maintainability and customization.

Linux also supports multi-user environments, providing strong access control mechanisms, user namespaces, and capabilities-based security, which helps isolate processes and enhance system security. Additionally, Linux integrates well with containerization technologies like Docker, which allows for the consistent and efficient deployment of applications across a variety of environments. These features make Linux highly suitable for hosting scalable and secure applications like *Draw It or Lose It* in a distributed architecture.

1. **Storage Management**: For storage management through the Linux OS, you have options such as Logical Volume Manager (LVM) and ext4 file system, which are server applications that are reliable and efficient. LVM is capable of dynamic resizing partitions and better disc space utilization, which is great for storage space management. The ext4 file system, known for its robustness and journaling capabilities, is widely used in production environments for maintaining data integrity and fast file access. For Draw It or Lose It, a hybrid approach would be an ideal solution for storage needs. While local storage using LVM and ext4 ensures high performance and control for critical system-level data, cloud-based storage solutions such as Amazon S3 or Google Cloud Storage would be well-suited for storing scalable, distributed content like game assets such as images, user profiles, and saved game progress. These services would provide redundancy, data backup, and global accessibility, which are essential for supporting thousands of concurrent users across multiple platforms.
2. **Memory Management**: Linux OS efficiently manages memory through a combination of techniques like paging, swapping and caching. The kernel’s Out-Of-Memory (OOM) killer ensures that critical processes remain running under memory pressure. Paging utilizes virtual memory ensuring stability and preventing processes from interfering with one another. Swapping transfers less-active data from the RAM to the disk, freeing up memory, a feature that is particularly valuable when managing multiple simultaneous game sessions. Caching process enhances file and disk access, reducing waiting times for the user. These processes allow for the web services to run both a stable and scalable application that is efficient under heavy workloads.
3. **Distributed Systems and Networks**: For *Draw It or Lose It* to work across multiple platforms, it will need to have a distributed system where the Linux OS Server is able to interact with the various client platforms. Utilizing RESTful APIs will help to support the HTTP operatives that will enable the clients to interact with the shared data. Controlling the load on the network will need to be a consideration, as well as having the possibility of offline modes and caching to minimize load on the server. Load balancing, CDN services, and container orchestration can ensure high availability and fault tolerance. Redundancy and failover strategies will protect against outages, and monitoring tools (i.e. Prometheus) will help manage connectivity and performance. Server health will need to be monitored, and the ability to be expanded upon to handle increased client base.

**Security**: A multi-layered strategy will be employed to protect user data across platforms. Linux offers strong security tools such as SELinux, firewall controls, and encrypted storage. The application will use HTTPS with TLS 1.3 for encrypted data transmission. OAuth2 will be implemented for user authentication and RBAC (Role-Based Access Control) for authorization. Platform-specific features like biometric authentication (fingerprint or facial recognition) will be supported on Android and iOS. Enforced session timeouts and IP filtering will further enhance security. Monitoring and logging via fail2ban and auditD will detect and prevent unauthorized access attempts.