

# 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 |
| --- | --- | --- | --- |
| 4.0 | 06/20/25 | Attila Bordan | Completed Recommendation section  Corrected the Development Tools/Mobile Devices section |

**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)

Draw It or Lose It currently exists only as an Android application; the new project extends that single-platform code base into a browser-accessible, hardware-agnostic experience so users on desktops, laptops, tablets, and smartphones can all play together.  
Because the client has limited in-house infrastructure knowledge, they have contracted CTS to design and provision the underlying operating platform (servers, operating systems, storage, and networking) before development begins.  
To ensure smooth gameplay across heterogeneous devices, the chosen platform must scale horizontally to accommodate unpredictable player spikes and vertically to handle processor- and memory-intensive image rendering.

## Requirements

Only one instance of the GameService class can exist at any given time. Game, team, and player must have unique identifiers to support the multi-team and multi-player functionality. Game, team, and player names must be unique per client request.

## [Design Constraints](#_2et92p0)

The solution moves from an on-device, single-user Android model to a distributed, client-server architecture. At least three separate environments (development, test, production) must be provisioned, each with identical OS and hardware images. Clue images will live on the server, which increases the storage demand beyond the current mobile storage. The web interface will generate more users so image rendering must be fast and seamless between the database and server. Migrating off the phone’s built-in account security demands a new authentication perhaps OAuth.

## [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 abstract Entity class is a base class that defines common attributes (id, name) and methods (getId, getName, toString) for other classes eliminating redundant code. The Game, Team, and Player classes inherit from the Entity class. The users interact with the concrete Game, Team, and Player classes without needing to understand the underlying logic and implementation. Private array lists in Game and Team classes other classes to modify the lists. A single getGame(id) method can return any subclass that might extend the Game class in future. Team iterate over its Player list, and Game over its Team list, fulfilling the client’s requirement that each game manage multiple teams and each team manage multiple players. GameService enforces a single in-memory instance, supporting the requirement that only one game instance can exist in memory at any given time.

**"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 can host web-based applications using Apache, Nginx, or Node.js environments, and supports Docker for containerization. However, macOS is not widely used as a production server OS due to licensing restrictions and hardware costs. It is best suited for development or staging environments | Linux is the most popular server platform due to its performance, stability, and open-source nature. It supports virtually all server-side technologies and frameworks (Node.js, Apache, Nginx, Docker, etc.). It is highly scalable and has no licensing cost, making it ideal for hosting web applications in a distributed environment (Red Hat, n.d.). | Windows Server supports hosting through IIS, .NET, and modern server frameworks. It is enterprise-friendly and supports a wide variety of backend technologies. However, Windows Server comes with licensing fees, and while powerful, it is often considered less cost-effective and less flexible than Linux (Microsoft, n.d.). | Mobile devices are not suitable for hosting web-based applications due to limited resources, security constraints, and lack of server-grade networking capabilities. They are strictly client platforms. |
| **Client Side** | Supporting modern browsers like Safari ensures compatibility on macOS. Development considerations include ensuring UI responsiveness and testing for OS-level integrations such as notifications and full screen capabilities. Apple hardware and developer tools are expensive, and expertise in macOS UI standards may be necessary. | Most modern browsers like Firefox and Chrome are fully supported on Linux, and the platform is commonly used by developers. Development costs are lower. Cross-platform browser compatibility must be verified. | Windows users primarily use Chrome or Edge. Windows provides broad compatibility and easy access for testing, but developers must ensure layout consistency across display settings and browser versions. Microsoft provides strong backward compatibility, which may increase complexity. | Ensuring responsive design and mobile-first development is essential. Testing on various screen sizes, network speeds, and touch-based interfaces is required. Time and cost increase when ensuring compatibility with both iOS and Android, as each has its own design guidelines and testing requirements. |
| **Development Tools** | macOS supports tools like Xcode, Visual Studio Code, and IntelliJ. It is the only platform that can develop native iOS apps using Xcode. It now even does a type check at compile-time instead of run-time (Apple, n.d.). However, macOS hardware is expensive, and licensing for Apple developer programs adds cost. | Linux supports a wide range of free, open-source IDEs (VS Code, Eclipse, JetBrains IDEs) and command-line tools. It is a preferred environment for backend, server, and open-source development. No licensing fees make it very cost-effective. | Windows supports powerful IDEs like Visual Studio, as well as cross-platform tools like VS Code. Many enterprise teams already use Windows-based environments. However, some open-source tools may require configuration for compatibility. | Mobile devices are not used for software development. Development for mobile platforms is done using macOS, Linux, or Windows.  Cross-platform frameworks like Flutter or React Native reduce duplication of effort (Flutter, n.d.).  Android development is supported on all major desktop platforms via Android Studio, while iOS development is only supported on macOS through Xcode (Apple, n.d.). Mobile devices are target platforms, not development environments. |

## 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**: It is recommended to deploy Draw It or Lose It on a Linux-based server environment, specifically using a distribution like Ubuntu Server LTS or Red Hat Enterprise Linux. Linux offers very good stability, scalability, and flexibility for hosting web-based distributed applications. Its open-source nature eliminates licensing fees, which is ideal for a company like The Gaming Room (Red Hat, n.d.). Most modern development tools and CI/CD pipelines are Linux-native or well-supported on Linux.
2. **Operating Systems Architectures**: Linux follows a monolithic kernel architecture, providing high performance and direct access to hardware while maintaining modularity through dynamically loadable kernel modules. It efficiently manages processes, memory, file systems, and I/O. Linux supports multi-user, multi-tasking environments with robust security through user permissions, and namespaces for isolation which is critical for running multiple game sessions concurrently (Silberschatz et al., 2009).
3. **Storage Management**: Draw It or Lose It requires storing approximately 1.6 GB of high-definition images (200 images × 8 MB each), along with player profiles, team data, and session states. A hybrid storage architecture is recommended or the application. Storing image assets in an object storage solution like Amazon S3 allowing fast, scalable access across platforms. Using a relational database such as PostgreSQL or MySQL for structured data user and game data.
4. **Memory Management**: To meet the requirement of rendering images at a fixed rate, the server should employ paging and demand paging techniques to avoid loading all assets into RAM. Linux supports virtual memory through the paging mechanism, swapping inactive pages to disk as needed. Memory-mapped files can also be used for efficient access to large image files. Preloading a small buffer of images for the next few game turns, while simultaneously clearing unused memory using garbage collection, ensures performance under load (Silberschatz et al., 2009).
5. **Distributed Systems and Networks**: Draw It or Lose It will operate as a distributed system, with clients connecting to a centralized server via HTTP from various web browsers and mobile devices. RESTful APIs will manage communication between frontend and backend components. For scalability and availability, a load balancer is recommended like Nginx to distribute traffic across multiple application instances. The application should be deployed using containers like Docker managed by Kubernetes to ensure seamless scaling and high availability. Implement monitoring and failover strategies to detect outages and reroute traffic accordingly.
6. **Security**: To protect user data and ensure secure gameplay HTTPS protocol is used that encrypts all communications between the server and the client. Implementing OAuth 2.0 for user authentication to improve security. Storing passwords according to best practices with hashing and salting. With the above techniques a secure, scalable and flexible environment will be created for cross-platform access. The setup allows future implementation of support of other platforms should the market conditions require it.

References

Apple. (n.d.-a). *Xcode overview and requirements*. Apple Developer. https://developer.apple.com/xcode/

Red Hat. (n.d.). *Why choose Red Hat for Linux?* https://www.redhat.com/en/topics/linux/why-choose-red-hat-enterprise-linux

Flutter. (n.d.). *Flutter for cross-platform development*. https://flutter.dev/

Microsoft. (n.d.). *Windows Server documentation*. https://learn.microsoft.com/en-us/windows-server/

Silberschatz, A., Galvin, P. B., & Gagne, G. (2008). *Operating system concepts* (8th ed.). Wiley.