

# **Draw It or Lose It**

# **CS 230 Project Software Design**

Alex Kastigar

Version 1.3

## 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 | 03/23/2025 | Alex Kastigar | Initial version of Project One software design document |
| 1.01 | 04/01/2025 | Alex Kastigar | Changed Document based on professor feedback and original assignment |
| 1.2 | 04/06/2025 | Alex Kastigar | Created Evaluation Table |
| 1.3 | 04/20/2025 | Alex Kastigar | Complied Research to Write Official Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is looking to expand its successful Android-based game, Draw It or Lose It, to operate on multiple platforms, including web and mobile. The client needs a scalable, browser-based application that supports thousands of users simultaneously. This solution must accommodate both mobile and desktop platforms while minimizing redundancy in code development. As a consultant, I am evaluating platforms and development tools that support these requirements in a cost-effective, maintainable, and secure way.

## Requirements

• The application must be web-based and run in modern web browsers.

• It must support both desktop (Windows, Mac, Linux) and mobile (Android, iOS) platforms.

• Thousands of users should be able to play simultaneously without performance degradation.

• Role-based user authentication is required to secure game access.

• The system must be easy to update and maintain across platforms.

## [Design Constraints](#_2et92p0)

1. Platform Scalability and Environment Requirements

Because the application is moving from a single-user mobile setup to a multi-user web-based system, it will require at least three dedicated environments: development, testing, and production. These environments need to support concurrent connections from thousands of users, which places constraints on the operating system and hardware choices. The server platform must support high scalability, meaning we’ll likely need to use a cloud-based solution that can scale compute and memory resources on demand. Additionally, testing environments may require duplication of services in isolated sandboxes, which could increase infrastructure costs and complexity. These requirements eliminate low-resource platforms and demand an OS like Linux or Windows Server with robust virtualization or container support.

1. Memory, Storage, and Image Rendering Constraints

Moving from a mobile-only app to a server-hosted game introduces new constraints around memory and storage. The stock images rendered for the game will be larger and must load at higher quality and speed to meet expectations across devices. This will require high-speed SSD storage and enough RAM on the server to pre-load and serve image assets without lag. Platforms with limited caching or slower I/O performance (like traditional HDDs or entry-level shared hosting) may no longer be suitable. This shift puts pressure on the chosen OS and hardware to support high-throughput operations, efficient memory management, and performance tuning capabilities.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram shows a domain model that follows core object-oriented programming principles to meet the specific needs of The Gaming Room's “Draw It or Lose It” application.

The Entity class acts as an abstract base class and introduces shared properties, id and name. These fields are inherited by the Game, Team, and Player classes, ensuring every game object has a consistent structure with a unique identifier and name. This satisfies the requirement that all three must have unique IDs and names.

Inheritance is used by having Game, Team, and Player extend the Entity class. This avoids repeating shared logic and keeps the code cleaner. The benefit is that we only have to define id, name, and related methods once, and they’re reused by all subclasses.

Abstraction is shown through the Entity class itself. It hides the internal implementation and acts as a simplified, generic concept that is extended by more specific classes like Game, Team, and Player.

Encapsulation is handled within each class by keeping properties like players and teams private and providing public methods (like addPlayer() and addTeam()) to access or modify them. This protects the internal state and makes it easier to manage data safely.

Polymorphism is applied through methods like getGame() in GameService, which returns a Game based on either ID or name. The method behavior can vary depending on what’s passed in or how it’s implemented, showing flexibility in how objects interact.

Composition is preferred over inheritance in the way Game has a list of Team objects and each Team has a list of Player objects. These relationships reflect real-world connections (a game has teams, and a team has players) and let us build more complex structures by combining objects.

The GameService class plays a critical role in managing games, teams, and players. It uses the singleton pattern, which ensures only one instance of the service exists in memory at a time. This directly supports the requirement that only one game instance should exist at any given moment.

To avoid duplicate entries when adding games, teams, or players, GameService uses the iterator pattern to scan its lists for matching names or IDs. This enforces uniqueness and satisfies the requirement that users must be warned if a name is already in use.

Lastly, classes like ProgramDriver and SingletonTester are not part of the game logic itself. They are used only for testing and demonstrating that the Singleton pattern works as expected, not as part of the production game design.

**"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)

This table compares four environments (Mac, Linux, Windows, and Mobile Devices) on hosting a web-based application (server side) and on client side considerations. Each cell addresses characteristics, advantages, weaknesses, and typical tools.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS can run web servers like Apache Tomcat or Nginx and supports Java applications. However, it's rarely used for production hosting due to high hardware costs, limited server market presence, and licensing constraints. Suitable for development or lightweight testing environments, but not recommended for large-scale deployment. | Linux is the industry standard for web-based application hosting. It's open-source, free to use, and highly customizable. It supports tools like Apache, Nginx, and Java servers (i.e. Tomcat, Jetty). It’s scalable, secure, and well-documented. Ideal for production environments (Linux Foundation, 2023). | Windows Server supports IIS, Tomcat, and Java environments. It's commonly used in enterprise settings. However, it comes with licensing costs and may require additional system resources. It's manageable for dev teams familiar with GUI-based administration (Microsoft, 2023a). | Mobile platforms like Android and iOS are not used to host server software. Their role is as clients only. Hosting server-side components on mobile devices is impractical due to limited resources and lack of persistent connections. |
| **Client Side** | Mac users can access the game through modern browsers like Safari or Chrome using a responsive web app. A consistent experience is possible as long as the app uses standard HTML5, CSS, and JavaScript. Java support is less common, so browser-based delivery is preferred over Java applets or desktop clients (Mozilla Developer Network, n.d.). | Linux desktops have a small user base but support Firefox and Chrome. A browser-based responsive design ensures compatibility. Java is well supported, but reliance on GUI-heavy interfaces should be avoided for wider compatibility (DigitalOcean, n.d.). | Windows has the largest desktop market share. The application must work across browsers like Edge, Chrome, and Firefox. As long as responsive design is used, the same client codebase can serve all desktop platforms. No extra tools are needed unless building a native client (not recommended for cost) (Stack Overflow, 2023). | The client experience must be responsive and touch-friendly. Android and iOS require testing across device sizes. A responsive web app is ideal for budget and code reuse. If performance becomes a concern, native apps can be developed, but at a higher cost and time investment (W3C, n.d.). |
| **Development Tools** | Supports Java development via IntelliJ, Eclipse, or NetBeans. Xcode is required for native iOS/macOS development. macOS is required to build iOS apps. Open-source tools help reduce cost, but Apple’s ecosystem requires developer licenses (Apple Developer, n.d.-b). | Ideal environment for server-side Java development using Eclipse, IntelliJ, Maven, and Gradle. Fully open-source with no licensing costs. Most CI/CD pipelines and cloud deployments run on Linux, so this aligns well with modern development practices (Linux Foundation, 2023). | Visual Studio Code and Eclipse run well on Windows. It supports Java, as well as .NET if the stack changes in the future. Some developer tools require licensing (like Microsoft SQL Server), but many Java tools are free. Familiar to devs in corporate environments (Microsoft, 2023b). | Android Studio is used for Android, and Xcode is required for iOS. Native app development requires platform-specific expertise and developer accounts (Apple Developer Program: $99/year). Web-first development using frameworks like React or Vue.js allows a unified approach with reduced duplication (JetBrains, 2023). |

Apple (macOS + iOS development)

Apple Developer. (n.d.). Xcode overview. Apple Developer. https://developer.apple.com/xcode/

Apple Inc. (n.d.). App distribution guide. Apple Developer. https://developer.apple.com/app-store/distribution/

Linux (Hosting + Development)

The Linux Foundation. (2023). Why Linux is perfect for web hosting. https://www.linuxfoundation.org/resources/publications/

DigitalOcean. (n.d.). Why Linux is ideal for developers. https://www.digitalocean.com/community/tutorials/why-use-linux-for-development

Microsoft (Windows Server + Dev Tools)

Microsoft. (2023). Windows Server 2022 overview. https://learn.microsoft.com/en-us/windows-server/get-started/whats-new-in-windows-server-2022

Microsoft. (n.d.). Visual Studio IDE overview. https://visualstudio.microsoft.com/vs/

General Platform Comparison & Cross-Platform Web Development

Mozilla Developer Network. (n.d.). Responsive design basics. https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\_layout/Responsive\_Design

Stack Overflow. (2023). Developer survey results 2023. https://survey.stackoverflow.co/2023/#most-popular-technologies

JetBrains. (2023). The state of developer ecosystems 2023. https://www.jetbrains.com/lp/devecosystem-2023/

W3C. (n.d.). HTML5 introduction. https://www.w3.org/TR/html5/

## Recommendations

For The Gaming Room’s expanded deployment of *Draw It or Lose It*, I recommend using a Linux-based cloud hosting platform, such as Ubuntu on AWS or Azure, as the primary server operating system. Linux is widely used in scalable web environments, supports modern development stacks, and comes with no licensing fees (HOSTAFRICA, 2023). Its performance, security, and configurability make it the industry standard for hosting large-scale web applications (Linux Foundation, 2016). Major cloud platforms like AWS and Azure offer pre-built Linux server instances with strong uptime, monitoring, and automated scaling capabilities (Amazon Web Services, n.d.).

Linux follows a modular monolithic kernel architecture, which offers direct performance benefits and efficient resource management (Canonical, 2023). Unlike Windows, which leans on graphical UI-heavy components, Linux can be configured to run lightweight headless server environments that dedicate more CPU and memory to application performance (IBM, 2022). This flexibility is key when hosting an image-heavy, multi-user application like *Draw It or Lose It*. With support for virtualization (via Docker, LXC) and multi-user environments, Linux is also well-suited to managing the testing, development, and production layers needed for this project (Red Hat, 2023).

To support 200+ high-resolution images (~8MB each) and user data (profiles, teams, scores, and match history), scalable object storage like Amazon S3 or Azure Blob Storage should be used (Amazon Web Services, n.d.; Microsoft, 2023). These storage services allow the application to offload static content (images) from the core server, freeing up memory and reducing I/O bottlenecks. They support versioning, backups, lifecycle rules, and access control for secure storage. File metadata can be managed in a relational database (SQL) running on the Linux server, while image content is retrieved through CDN-backed storage for fast delivery (Linux Foundation, 2023).

Linux offers excellent memory management through paging, caching, and buffer tuning (IBM, 2022). For a game that renders images in real time, memory optimization is critical. Using in-memory caching systems such as Redis or Memcached allows the application to store frequently used image data and game session info for quick access, reducing image loading latency (DigitalOcean, n.d.). The Linux kernel supports memory overcommit policies and swap management to prevent crashes during high loads (Canonical, 2023). Monitoring tools like *top*, *htop*, or *vmstat* help track and optimize real-time memory performance (ElGazzar, 2023).

To serve users across desktop and mobile platforms, *Draw It or Lose It* should use a distributed microservices architecture. This allows different parts of the application (authentication, game logic, chat, image service, etc.) to run in isolated containers, deployed across a cluster (Red Hat, 2023). Using Docker with Kubernetes (The Kubernetes Authors) or ECS provides orchestration and auto-scaling. RESTful APIs enable communication between services and clients over HTTPS, while WebSockets can be used for real-time multiplayer interactions (Mozilla, n.d.-b). Cloud-based load balancers and reverse proxies ensure uptime even if some nodes fail, and CDN integration improves global access speeds (Linux Foundation, 2016).

Security should be layered across all parts of the system. User credentials and game data must be encrypted both in transit (TLS 1.2+) and at rest. HTTPS will be enforced site-wide using free SSL certificates (Let’s Encrypt) (Mozilla, n.d.-a). On the back end, Linux firewalls (*iptables* or *ufw*) restrict access to internal ports (IBM, 2022). Role-based access control (RBAC) and API authentication (OAuth 2.0 or JWT) will ensure users only access permitted features. Distributed denial-of-service (DDoS) protections via the cloud provider (like AWS Shield or Azure DDoS Protection) will already protect the application on the backend (Amazon Web Services, n.d.).

## References

Amazon Web Services. (n.d.). Amazon S3. Retrieved from https://aws.amazon.com/s3/

Microsoft. (2023). Azure Blob Storage. Retrieved from https://azure.microsoft.com/en-us/products/storage/blobs/

Linux Foundation. (2023). What is Linux. Retrieved from https://www.linuxfoundation.org/resources/publications/

Mozilla. (n.d.-a). Introduction to HTTPS. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

DigitalOcean. (n.d.). How to use Redis for caching. Retrieved from https://www.digitalocean.com/community/developer-center/caching-using-digitalocean-redis-on-app-platform

IBM. (2022). *Advantages of Linux over Windows*. Retrieved from https://www.ibm.com/docs/en/linuxonibm

Red Hat. (2023). *Security and compliance with open-source systems*. Retrieved from https://www.redhat.com/en/topics/security

HOSTAFRICA. (2023, August 10). *Linux vs Windows – TCO, ROI and more*. HOSTAFRICA. https://www.hostafrica.com/blog/linux/linux-vs-windows-tco-roi-and-more/

Canonical. (2023, February 14). *Real-time Ubuntu is now generally available*. Canonical. https://canonical.com/blog/real-time-ubuntu-is-now-generally-available

The Linux Foundation. (2016, August 18). *LinuxCon preview: Marten Mickos on why Linux dominates in cloud*. Linux Foundation. https://www.linuxfoundation.org/blog/blog/linuxcon-preview-marten-mickos-on-why-linux-dominates-in-cloud

ElGazzar, K. (2023, April 14). *The essential guide to Linux system monitoring with top, htop, and vmstat*. SWAC. https://swac.blog/the-essential-guide-to-linux-system-monitoring-with-top-htop-and-vmstat/#:~:text=While%20top%20and%20htop%20provide,view%20of%20your%20system's%20performance.

The Kubernetes Authors. (n.d.). *Kubernetes*. Kubernetes.io. https://kubernetes.io/

Mozilla. (n.d.-b). *WebSockets API*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API