

# Draw It or Lose It Web-Based Game

# **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 | 07/17/2025 | Vanessa Sanchez | Initial version of software design draft |
| 1.1 | 07/31/2025 | Vanessa Sanchez | Revised Project One based on instructor feedback; corrected design and code structure |
| 1.2 | 08/01/2025 | Vanessa Sanchez | Added Project Two platform evaluation section |
| 1.3 | 08/13/2025 | Vanessa Sanchez | Revised Project Two based on instructor feedback;  Corrected evaluation section and added references |
| 1.4 | 08/15/2025 | Vanessa Sanchez | Added Project Three platform recommendation 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)

The Gaming Room wants to take their current Android game, *Draw It or Lose It*, turned it into a web-based game that works across different platforms. Right now, their team doesn’t have the technical experience to set up that kind of environment, so they’ve asked us to help.

To solve this, I’m proposing a design that uses object-oriented programming and common design patterns like singleton and iterator. These will help us meet their requirements such as making sure there’s only one version of the game running at a time and that all game and team names are unique. This setup will make the game easier to manage and update in the future, and it’ll also support different devices and platforms.

## Requirements

* The game should allow one or more teams to play.
* Each team should have multiple players.
* Game and team names must be unique.
* There should only be one instance of the game running in memory.

## [Design Constraints](#_2et92p0)

One major design constraint is that the project must support three or more distinct environments which are development, testing, and production. Each will require dedicated infrastructure and platform configurations. For example, testing may be divided into QA and UAT environments that require different servers or containers. These environments must scale as needed to support additional users, making a cloud-based platform like AWS or Azure more practical than a static, on-premises setup.

Another design constraint is the shift from a single-user mobile app to a multi-user web application. Unlike the current setup that uses a device's native security, the web version needs robust user authentication and session management systems. This includes creating login functionality, ensuring secure token-based access, and making sure no user is logged in from multiple devices at the same time.

## [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)

Designing the domain model helped me understand how object-oriented programming supports the client's requirements. I used inheritance by creating a base Entity class for shared attributes like ID and name, which Game, Team, and Player all inherit. This ensures every object is uniquely identified, meeting the need for unique names and IDs.

Encapsulation is used by keeping variables private and using getters and setters to access them. Abstraction helps by moving common features into the Entity class, which simplifies the structure. Polymorphism allows each class to customize behavior when needed by overriding methods.

To ensure only one game runs at a time, I used the Singleton pattern. I also used the Iterator pattern to check for duplicate names, supporting name uniqueness. These patterns help keep the system organized and meet all client requirements.

**"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** | Mac is UNIX-based but needs expensive licensing for servers (Red Hat, 2023). It supports Docker but isn't great for big server setups. The virtualization isn't as good as Linux, and cloud options are limited. LDAP setup takes more work. It's okay for small projects but gets too costly for bigger apps. | Linux is the best choice for servers because it's open-source and free (Linux Foundation, 2023). It's UNIX-based, stable, and handles virtualization well with Docker and Kubernetes. Works great with all cloud platforms like AWS and Azure. LDAP integration is easy, and security controls are excellent. Most web servers already use Linux. | Windows Server costs a lot in licensing fees but is familiar to use (Microsoft, 2023). It's not UNIX-based but supports Hyper-V and Docker. Works well with Microsoft's cloud and Active Directory. Uses more resources than Linux though, which means higher costs. Good for companies already using Microsoft stuff. | Mobile devices can't work as servers (Apple Developer, 2023). Even though iOS is UNIX-based and Android is Linux-based, they're made for apps, not hosting. They can't handle multiple connections or server tasks. Mobile devices are just for running the client side of the app. |
| **Client Side** | Mac supports Safari, Chrome, and Firefox browsers (Apple, 2023). Safari works best on Apple devices, but Chrome is more popular overall. About 27% of people use Mac desktops. Good for development with Safari tools but focusing only on Safari limits your audience since Chrome is used more. | Linux supports Chrome, Firefox, and Edge browsers (Ubuntu, 2023). Only about 3% of people use Linux desktops, but developers like it. Chrome works great on Linux. The small user base means less priority from browser companies, but it's good for security and privacy. | Windows has the biggest user base at about 70% of desktops (StatCounter, 2023). Supports all major browsers including Chrome (most popular), Edge, and Firefox. Since most people use Windows, you get the largest potential audience. Development tools like Visual Studio are excellent for web apps. | Mobile is super important now - more people browse on phones than computers (Statista, 2023). Android and iOS make up almost 100% of mobile. The app must work on different screen sizes. Mobile browsers like Chrome Mobile and Safari Mobile can run games. Need responsive design and maybe Progressive Web Apps for better mobile experience. |
| **Development Tools** | Mac has Xcode for iOS apps and supports tools like VS Code and Eclipse (Apple Developer, 2023). You need Mac hardware for iOS development though. Good terminal access and tools like Homebrew. Docker works well. But you're stuck with Apple hardware if you want to make iOS apps. | Linux is great for development with support for all programming languages (Linux Foundation, 2023). Package managers make installing tools easy. Docker and Kubernetes work perfectly. IDEs like VS Code and Eclipse run great. Best command-line tools and lots of open-source options for web development. | Windows is familiar to most developers and has excellent tools like Visual Studio (Microsoft, 2023). Great .NET support and works well with Microsoft services. Docker Desktop and WSL let you work with Linux-like environments. Some open-source tools need extra setup, but overall, it's the easiest platform for most developers. | You don't develop ON mobile devices - you develop FOR them using the other platforms (Google Developer, 2023). Use Android Studio for Android apps and Xcode for iOS apps. Tools like React Native let you code once for both platforms. Need simulators and real devices for testing. Mobile devices are just targets, not development machines. |

## 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 Ubuntu Server as the main operating platform for expanding Draw It or Lose It to other computing environments. Linux is the best choice because it's open-source and free, which saves money on licensing costs (Red Hat, 2023). Since it's UNIX-based like macOS and iOS, it'll work well with different platforms. Linux runs on over 90% of web servers worldwide, so there's a lot of support and documentation available. It also handles multiple users and processes well, which is perfect for a multiplayer game where many people will be playing at the same time.

**2. Operating Systems Architectures**

Linux uses a monolithic kernel architecture with loadable modules, which gives us good performance and flexibility (Linux Foundation, 2023). The kernel handles all the main functions like process management, memory management, and device drivers in one place, making it fast. But we can still add modules when we need new features without restarting the system. Linux supports preemptive multitasking, meaning it can switch between different game sessions quickly and fairly. It also has multi-user capabilities built in, so multiple players can access the system safely at the same time. The hierarchical file system organizes everything clearly, and the process scheduling makes sure that no single game session uses all the resources.

**3. Storage Management**

For storage management, I recommend using a hybrid approach with both local and cloud storage (Amazon Web Services, 2023). We'll use the ext4 file system on the local Linux servers because it's reliable and handles large files well. For the game data, we should use Amazon RDS for the database information like user accounts, game scores, and team details. For storing game images and media files, Amazon S3 object storage works great because it automatically backs up files and can handle many users downloading images at once. This setup also gives us automatic backups and data replication across multiple locations, so if one server goes down, the game keeps working.

**4. Memory Management**

Linux uses several memory management techniques that work well for Draw It or Lose It (Torvalds et al., 2023). It uses virtual memory to give each game session its own memory space, so one crashed game won't affect others. Demand paging means the system only loads the parts of the game that are being used, saving memory. The page replacement algorithms like LRU (Least Recently Used) automatically free up memory from inactive game sessions. Linux also uses memory pools and buffer caches to speed things up by keeping frequently used game data in memory. For our multiplayer game, we can use connection pooling to reuse database connections and session management to track active players without wasting memory.

**5. Distributed Systems and Networks**

To make Draw It or Lose It work across different platforms, I recommend using a microservices architecture with containerization (Fowler, 2023). We'll break the game into smaller services like user authentication, game logic, team management, and real-time communication. Each service runs in its own Docker container, making it easy to update and scale individual parts. We'll use load balancers to distribute incoming players across multiple servers, and message queues like RabbitMQ to handle real-time updates between players. API gateways will provide a single-entry point for mobile apps, web browsers, and other clients. For handling network dependencies, we need circuit breakers that can detect when a service is down and route traffic elsewhere. Health checks will monitor all services and retry mechanisms will handle temporary connection issues. This setup means if one component fails, the others keep working, and players on different platforms can still play together.

**6. Security**

Security is important for protecting user information across all platforms (OWASP, 2023). We'll use HTTPS/TLS encryption for all communication between clients and servers, so nobody can intercept game data or passwords. OAuth 2.0 will handle user login securely, and JWT tokens will manage user sessions with automatic expiration. Linux provides built-in security with iptables firewall, access control lists (ACLs), and user permissions to control who can access what. We'll implement input validation to prevent injection attacks, rate limiting to stop DDoS attacks, and CSRF tokens to prevent cross-site attacks. For data protection, we'll use database encryption for stored user information and password hashing with bcrypt. Regular security updates and automated vulnerability scanning will keep the system protected. We'll also use intrusion detection systems to monitor for suspicious activity and audit logging to track all system access.

**References**

Amazon Web Services. (2023). *AWS cloud storage solutions*. <https://aws.amazon.com/storage/>

Android Developer. (2023). *Android development guidelines*. <https://developer.android.com/>

Apple. (2023). *macOS developer documentation*. <https://developer.apple.com/macos/>

Apple Developer. (2023). *iOS development guide*. <https://developer.apple.com/ios/>

Fowler, M. (2023). *Microservices architecture patterns*. <https://martinfowler.com/microservices/>

Google Developer. (2023). *Android Studio development environment*. <https://developer.android.com/studio>

Linux Foundation. (2023). *Linux kernel development and architecture*. <https://www.kernel.org/>

Microsoft. (2023). *Windows Server development platform*. <https://docs.microsoft.com/windows-server/>

OWASP. (2023). *Web application security guidelines*. <https://owasp.org/>

Red Hat. (2023). *Enterprise Linux server solutions*. <https://www.redhat.com/en/technologies/linux-platforms>

StatCounter. (2023). *Global desktop operating system market share*. <https://gs.statcounter.com/>

Statista. (2023). *Mobile vs desktop internet usage statistics*. <https://www.statista.com/topics/779/mobile-internet/>

Torvalds, L., et al. (2023). *Linux kernel memory management*. <https://www.kernel.org/doc/html/latest/vm/>

Ubuntu. (2023). *Ubuntu desktop environment documentation*. <https://ubuntu.com/desktop>

<https://ubuntu.com/desktop>