

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_eu0vgikn2xuh)

[**Table of Contents 2**](#_twt26kghded)

[**Document Revision History 2**](#_5bw5vad8htu0)

[**Executive Summary 3**](#_2a5ktzwb670c)

[**Requirements 3**](#_gy225yoybczb)

[**Design Constraints 3**](#_wi62s63x1izz)

[**System Architecture View 3**](#_bcnvyj7iyd20)

[**Domain Model 3**](#_4e0m5978qqnu)

[**Evaluation 4**](#_cr5ky5cf39k7)

[**Recommendations 5**](#_2etc44nek38e)

## [Document Revision History](#_bq2q8lyked36)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 08/03/2025 | Eriik Snyder | Completed the Evaluation section with a detailed analysis of Mac, Linux, Windows, and Mobile platforms based on server-side capabilities, client-side development considerations, and relevant development tools. |

**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](#_nqezy0wy2qzf)

## Requirements

The Gaming Room has requested an evaluation of platform options to expand their Android-only game application, Draw It or Lose It, into a web-based application that can serve users across various desktop and mobile platforms. This expansion requires analyzing the feasibility, advantages, and limitations of deploying the game on Mac, Linux, Windows, and mobile platforms, both on the server and client sides. The core challenge lies in creating a responsive and scalable web application capable of supporting thousands of concurrent players, while ensuring compatibility across devices and operating systems. This document presents a detailed evaluation of each platform's development considerations and technical requirements. The proposed solution includes deploying the game as a web application using open-source tools where possible, leveraging responsive HTML5 for cross-platform support, and prioritizing server environments that balance cost, scalability, and developer familiarity. These findings will support The Gaming Room in making an informed decision for their expansion strategy.

## [Design Constraints](#_kflom9pg9ih3)

Developing Draw It or Lose It as a web-based application in a distributed environment introduces several key design constraints. First, the application must support high concurrency, as multiple teams may play simultaneously. This requires a scalable backend architecture, likely using a RESTful API and cloud-based hosting to handle load balancing and failover. Second, the frontend must be fully responsive and compatible with a variety of browsers and devices, meaning HTML5, CSS3, and JavaScript frameworks must be used with rigorous cross-platform testing. Third, mobile platforms (iOS and Android) may need hybrid or native wrappers for optimal performance and App Store deployment, which adds complexity. Finally, security is critical, especially in handling user sessions and personal data, requiring secure authentication, HTTPS communication, and regular security audits. These constraints affect the development timeline, required expertise, and tool selection, and they will likely necessitate multiple development environments and testing stages to ensure seamless integration across all supported platforms.

## [System Architecture View](#_jr051cspfk9e)

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](#_qscqvu7new43)

The UML class diagram outlines the core object relationships within Draw It or Lose It. The Game class serves as the main controller, coordinating gameplay between multiple Team objects. Each Team is composed of Player objects, allowing for team-based interactions. An abstract Entity class may serve as a parent to shared attributes or methods between Game, Team, and Player, promoting code reuse and consistency. This design demonstrates several key object-oriented programming (OOP) principles. Encapsulation is applied by defining class responsibilities and hiding internal data. Inheritance is present if Entity is used as a base class, reducing redundancy. Polymorphism allows for flexible object behavior through shared interfaces or base class methods. Abstraction is achieved by modeling only the relevant properties and behaviors needed to support the game's structure. Together, these principles enable a modular, maintainable design that supports scalability and reusability in a multi-platform web-based environment.

**"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](#_vynqtphfwgxk)

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 hosting web applications through tools like Apache, Nginx, and Docker, but it is rarely used in production environments due to licensing costs and limited server optimization. macOS Server is deprecated, and deployment at scale on Mac is impractical. | Linux is the most popular and cost-effective server OS for hosting web-based applications. It offers excellent performance, scalability, open-source tools, and robust community support. Linux distributions like Ubuntu Server are ideal for running Apache, Nginx, or Node.js servers. | Windows Server can effectively host web applications, particularly with Microsoft-based tech stacks (IIS, ASP.NET). It offers GUI-based management and enterprise-level support, but licensing costs can be high compared to Linux. | Mobile devices are not suited for hosting server applications. They are designed as client endpoints, not persistent servers. While mobile apps can create local servers for temporary use (e.g., during development), they lack the capacity, networking control, and uptime required for production hosting. |
| **Client Side** | To support Mac users, the web app must be compatible with Safari and Chromium-based browsers. Responsive HTML5 and cross-browser testing are essential. Costs and expertise are moderate; macOS offers great developer tools, but testing on Apple hardware is necessary. | Linux desktop users may use browsers like Firefox and Chrome. Since Linux distributions vary, testing across versions is important. Costs are low, and development tools are open-source, but support and browser compatibility should be verified. | Windows dominates the desktop OS market. Ensuring compatibility with Microsoft Edge, Chrome, and Firefox is essential. Most tools work natively on Windows, and a wide range of testing frameworks are available. Expertise is widely available. | Supporting both iOS and Android requires responsive design and extensive mobile browser testing. Tools like React Native or Flutter may reduce cost and effort for native versions. However, careful testing across device sizes and operating systems is crucial. |
| **Development Tools** | macOS supports a wide range of tools, including Xcode, Visual Studio Code, and IntelliJ. It’s required for iOS development, making it essential if mobile expansion is a priority. Costs may include Apple hardware and licensing for App Store deployment. | Linux supports robust development environments such as Eclipse, VS Code, and JetBrains IDEs. It’s ideal for backend and web development using Node.js, Python, and Java. Most tools are free and open-source. Great for budget-conscious teams. | Windows supports Visual Studio, Unity, and other major IDEs. It’s compatible with most tools and languages and ideal for C# and .NET development. Licensing may be required for Visual Studio Professional, but open-source alternatives exist. | Android development is best done using Android Studio. iOS development requires Xcode on macOS. Cross-platform tools like Flutter, React Native, and Xamarin can reduce duplication. Testing frameworks and simulators are necessary. Some tools have licensing costs. |

## 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**: The recommended operating platform for expanding Draw It or Lose It is Linux, specifically using a distribution such as Ubuntu Server. Linux provides a stable, scalable, and cost-effective environment for hosting web-based applications. It supports a wide range of open-source tools, including Apache, Nginx, Docker, and Node.js, which are ideal for deploying distributed applications. Linux is widely used in cloud hosting environments such as AWS, Azure, and Google Cloud Platform, making it well-suited for scaling the game to support thousands of concurrent players. Its reliability, robust security features, and strong community support make Linux the optimal choice for The Gaming Room to expand into multiple computing environments while maintaining flexibility and performance.
2. **Operating Systems Architectures**: Linux supports a modular and layered architecture that makes it highly customizable and efficient for server-side deployments. At its core, the Linux kernel handles process management, memory management, and device drivers. On top of the kernel, user-space applications, libraries, and services run independently, allowing for precise control over system resources. This architecture enables developers to deploy only the necessary components, reducing overhead and improving performance. Linux also supports multi-user and multitasking environments, making it ideal for web applications that serve many users simultaneously. Its compatibility with modern web servers, containers, and package managers allows for seamless integration with CI/CD pipelines and distributed microservices architectures. This flexibility ensures that Draw It or Lose It can be deployed securely, efficiently, and at scale across a wide range of hosting environments.
3. **Storage Management**: For the Linux-based deployment of Draw It or Lose It, the recommended storage management system is a combination of ext4 file system for local storage and cloud-based object storage such as Amazon S3 or Google Cloud Storage for scalability and redundancy. The ext4 file system is reliable, efficient, and well-supported in Linux environments, making it ideal for hosting application files and logs. For dynamic data such as game assets, user sessions, or backups, integrating cloud object storage ensures high availability, data durability, and easy integration with content delivery networks (CDNs). This hybrid approach allows The Gaming Room to scale storage resources on demand while maintaining fast local access to critical components, ensuring optimal performance and disaster recovery capabilities.
4. **Memory Management**: Linux uses an advanced memory management system that ensures efficient use of system resources, which is essential for running Draw It or Lose It in a distributed web-based environment. It employs virtual memory to provide each process with its own address space, improving stability and preventing conflicts. Linux uses demand paging and lazy loading to load only necessary parts of a program into memory, reducing overhead. Additionally, the kernel’s out-of-memory (OOM) killer prevents the system from becoming unresponsive by terminating low-priority processes when memory is critically low. For server-side scalability, Linux supports memory allocation through cgroups and namespaces, allowing containerized deployments (via Docker) to isolate memory use per instance. This ensures that the game application performs reliably under load and that resources are efficiently allocated across multiple concurrent sessions.
5. **Distributed Systems and Networks**: To support communication between various platforms, Draw It or Lose It should be built using a distributed software architecture based on RESTful APIs and hosted on a cloud-based platform (such as AWS or Azure). The application will consist of independent client and server components that communicate over the internet using HTTPS. This enables devices running Windows, macOS, Linux, Android, and iOS to access the game consistently via web browsers. On the backend, the server can use load balancers and horizontal scaling to distribute requests across multiple application instances, ensuring consistent performance and uptime.

To maintain resilience, redundancy and failover mechanisms must be in place. This includes using multiple availability zones, automatic backups, and health checks to detect and recover from connectivity failures. Dependencies such as authentication services, database connections, and content delivery networks must be managed to prevent single points of failure. Tools like Docker and Kubernetes can help isolate and manage services across nodes, ensuring smooth communication between components and high availability even during network interruptions or maintenance windows.

1. **Security**: Protecting user information across platforms is essential for Draw It or Lose It. On the Linux server platform, security is enforced through multiple layers. First, HTTPS encryption should be used to secure all client-server communications using SSL/TLS certificates. Linux supports robust firewall configurations (e.g., iptables or ufw) to restrict unauthorized access. For authentication, implementing secure methods such as OAuth 2.0 or JWT (JSON Web Tokens) will help ensure that only verified users can access game sessions.

User data should be stored using encrypted databases, and sensitive information (e.g., passwords) must be hashed using secure algorithms like bcrypt. Regular security patches, intrusion detection systems, and access control policies help prevent breaches on the server side. When dealing with mobile and desktop platforms, secure browser practices, such as same-origin policy and content security policies, should be applied to prevent cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks.

Additionally, containerization (e.g., Docker) offers further isolation between services, limiting the potential impact of compromised components. These combined strategies ensure data confidentiality, integrity, and availability across all platforms.