

Draw it or Lose it Web App

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design** 1](#_Toc200973519)

[Table of Contents 2](#_Toc200973520)

[Document Revision History 2](#_Toc200973521)

[Executive Summary 3](#_Toc200973522)

[Requirements 3](#_Toc200973523)

[Design Constraints 3](#_Toc200973524)

[System Architecture View 3](#_Toc200973525)

[Domain Model 3](#_Toc200973526)

[Evaluation 4](#_Toc200973527)

[Recommendations 7](#_Toc200973528)

[References 9](#_Toc200973529)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 06/16/2025 | Jessica Livingston | Submission for CS 230 Project Three |

**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 has tasked Creative Technology Solutions (CTS) with transitioning their existing Android-based game, Draw It or Lose It, into a web-based application accessible across multiple platforms. The core objective is to maintain game logic and structure while introducing scalability and support for team-based gameplay with unique identifiers. This document outlines the design constraints, object-oriented structure, and platform evaluations necessary to meet the client's software requirements and enable a successful deployment.

## Requirements

* *A game can support one or more teams.*
* *Each team can have multiple players.*
* *Game and team names must be unique.*
* *Only one instance of the game exists in memory at any time.*
* *Entities such as Game, Team, and Player must have unique identifiers and names.*

## [Design Constraints](#_2et92p0)

Developing the game in a web-based distributed environment introduces the following constraints:

* **Singleton Constraint**: Only one instance of the game should exist in memory. We will implement the Singleton design pattern for the GameService class to manage global access to game data.
* **Uniqueness Constraint**: All names (game, team, player) must be unique. We will use the Iterator design pattern to check name uniqueness during creation.
* **Scalability**: The app must support cross-platform functionality, impacting how we handle session management, network latency, and user state.
* **Concurrency**: The game must be thread-safe to handle simultaneous player interactions.
* **Portability**: All code must remain platform-agnostic and modular to accommodate future deployment across desktop and mobile browsers.

## [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 UML class diagram uses object-oriented principles to create a flexible and scalable model. The Entity class serves as a base with common attributes: id and name. Game, Team, and Player all extend from Entity, supporting inheritance and code reuse.

* GameService: Implements the Singleton pattern and manages creation and retrieval of games, teams, and players.
* Game: Holds references to one or more Team instances.
* Team: Holds references to multiple Player instances.
* Player: Represents an individual participant on a team.

These classes reflect encapsulation, inheritance, and abstraction, ensuring separation of concerns and extensibility.

**"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 supports local web server setups (e.g., Apache, NGINX) and development tools like Docker, but is rarely used in enterprise production due to higher hardware costs, limited server-grade support, and Apple’s licensing constraints. | Linux is the industry standard for server environments, offering excellent performance, security, and scalability. It supports robust web stacks (Apache, NGINX, Node.js), is cost-efficient, and integrates easily with DevOps pipelines and container orchestration tools. | Windows Server supports enterprise web hosting with tools like IIS and Azure integration. It offers GUI-based management but has higher licensing costs and greater resource overhead compared to Linux. Often chosen for compatibility with legacy systems. | Mobile devices (iOS, Android) are not suitable for server hosting. Their OSs are optimized for client-side performance, lack root access for server-level software, and are restricted by app store policies. |
| **Client Side** | macOS supports all modern browsers (Safari, Chrome, Firefox). However, Safari has some rendering quirks developers must test for. Ideal for frontend validation on Apple hardware. | Linux desktops support Chrome and Firefox, offering broad standards compliance. Though less common among average users, Linux testing ensures accessibility for open-source enthusiasts and developers. | Windows dominates desktop usage, requiring extensive client testing for performance, display scaling, and compatibility across browsers and Windows versions. | iOS and Android require responsive design, touch input optimization, and screen-size testing. Each platform has unique UX patterns, making cross-platform UI consistency a challenge but critical for reach. |
| **Development Tools** | macOS supports Xcode (mandatory for iOS development), IntelliJ, Eclipse, and VS Code. It is required for native iOS testing, but Apple licensing and hardware costs are restrictive. | Linux offers full support for open-source IDEs (VS Code, Eclipse), command-line tooling, Docker, and CI/CD pipelines. Ideal for full-stack and backend developers. Highly customizable for advanced workflows. | Windows supports Visual Studio (robust but resource-intensive), IntelliJ, and Eclipse. Strong enterprise ecosystem but some tools may incur licensing fees. Windows Subsystem for Linux (WSL) adds Linux compatibility. | Android Studio (for Android) and Xcode (for iOS) are essential for native development. Cross-platform frameworks like Flutter, React Native, or Unity allow a single codebase, but testing on real devices is necessary for accuracy. |

Summary

Based on the above evaluation, Linux is the most cost-effective and scalable platform for server-side deployment. Cross-platform responsive web development supported by open-source tools allows broad accessibility across desktop and mobile clients. Additional tooling is necessary for mobile responsiveness and iOS/Android compatibility.

## 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 that Draw It or Lose It be deployed on a Linux-based server platform, such as Ubuntu Server. Linux is widely adopted in enterprise and cloud-hosted environments due to its open-source licensing, low operational cost, high stability, and superior security features. It also supports essential web technologies such as Apache, NGINX, Node.js, and containerization tools like Docker, making it highly suitable for scalable web applications. Its compatibility with modern development pipelines ensures long-term support and flexibility as The Gaming Room expands its reach.
2. **Operating Systems Architectures**: The Linux platform employs a monolithic kernel architecture, which combines core system services like memory management, file system control, and device drivers into a single kernel space. This results in faster system calls and efficient inter-process communication, ideal for real-time gaming. The system will follow a client-server model, where the server manages game logic and persistent state, and the clients—using web interfaces—communicate with the server via RESTful APIs. This decoupled structure allows the system to remain scalable, fault-tolerant, and easy to maintain.
3. **Storage Management**: A combination of SSD-backed file storage and a relational database system is recommended to handle both static assets and dynamic game data. MySQL or PostgreSQL can be used to store structured data like user credentials, game sessions, teams, and player statistics, while cloud object storage (e.g., AWS S3) can be used for multimedia files. The Linux platform natively supports file systems like ext4, known for their stability and journaling capabilities, ensuring data integrity and fast read/write performance.
4. **Memory Management**: Linux supports virtual memory with paging and swap space, enabling efficient memory allocation and process isolation. Additionally, using the Java Virtual Machine (JVM) provides automatic garbage collection, preventing memory leaks and improving runtime efficiency. The implementation of the Singleton pattern ensures that only one instance of critical classes (like GameService) exists in memory, which aligns with the requirement to restrict the application to a single active game instance. This setup minimizes memory overhead and supports concurrent user access in a thread-safe manner.
5. **Distributed Systems and Networks**: To enable communication across platforms (desktop, mobile, etc.), the system will use distributed architecture techniques supported by HTTP/HTTPS protocols and WebSocket connections for real-time interaction. These components will be orchestrated using containerized services deployed on platforms such as Kubernetes, enabling horizontal scaling and fault tolerance. Dependency management will include implementing retries, fallback logic, and data synchronization strategies to handle outages, connectivity issues, and client-server state consistency.
6. **Security**: Security will be integrated at all layers of the application. Data in transit will be encrypted using TLS/SSL certificates. User authentication will be implemented using JSON Web Tokens (JWT), supporting secure, stateless sessions. Additionally, input validation and output sanitization will defend against SQL injection, XSS, and CSRF attacks. On the Linux platform, features such as SELinux/AppArmor and firewall rules (e.g., iptables/ufw) will further harden the server environment. Sensitive information, including user credentials, will be hashed using secure algorithms (e.g., bcrypt) and stored using encrypted volumes or database encryption at rest.

## References

Apache Software Foundation. (n.d.). Apache HTTP Server Project. https://httpd.apache.org/

Docker, Inc. (n.d.). What is Docker? https://www.docker.com/resources/what-container/

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

MySQL. (n.d.). MySQL: The world's most popular open source database. https://www.mysql.com/

Oracle. (n.d.). Java Platform, Standard Edition Documentation. https://docs.oracle.com/javase/8/docs/

SQLite. (n.d.). About SQLite. https://www.sqlite.org/about.html

Red Hat. (n.d.). What is Linux? https://www.redhat.com/en/topics/linux/what-is-linux

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

OWASP Foundation. (n.d.). OWASP Top Ten Web Application Security Risks. https://owasp.org/www-project-top-ten/

W3C. (n.d.). AJAX: Asynchronous JavaScript and XML. https://www.w3.org/TR/xmlhttpreq/

JSON Web Token (JWT). (n.d.). Introduction to JSON Web Tokens. https://jwt.io/introduction/

Kubernetes. (n.d.). Production-Grade Container Orchestration. https://kubernetes.io/

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

Google Developers. (n.d.). Android Studio User Guide. https://developer.android.com/studio/intro

React Native. (n.d.). React Native Documentation. https://reactnative.dev/