THE GAMING ROOM

Draw It or Lose It

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	<09/27/25>	Adrianne Scott-Brown	Development of Executive Summary, Requirements, Design Constraints, System Architecture View, Domain Model, Evaluation, Recommendations
1.0	10/4/25	Adrianne Scott-Brown	Evaluation, Recommendations, Implementation Strategy, Conclusion
1.1	10/16/25	Adrianne Scott-Brown	Recommendations, conclusion

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

The Gaming Room is looking to expand its successful Android-only game into a scalable, web-based platform. Currently, the game consists of four timed rounds, where teams render images and compete to correctly identify them. The competitive and time-sensitive structure requires a system that ensures performance, accuracy, and fairness across multiple users.
To meet these requirements, The Client Technology Solutions (CTS) team will be developing an object-oriented Java application that emphasizes scalability by enabling multiple teams and players to participate simultaneously while enforcing unique team and player names. By implementing a single game instance in memory, the architecture aims to reduce redundancy, ensure consistency, and optimize resource management.
The design is expected to incorporate established software architecture principles such as modularity, maintainability, and security. The application will be designed for web deployment, ensuring accessibility across platforms while preserving game integrity. Considerations for concurrency, data management, and future scalability are built into the solution to accommodate growth and evolving requirements.
This document outlines the design and development approach, providing a framework that supports both the technical requirements of the client and the long-term maintainability of the system.

Requirements

Requirement	***Type***	***Description***
Single game instance	*Functional*	*One game runs in memory at a time.*
Unique team & player names	*Functional*	*Will prevent duplicate names for teams or players.*
Multi-team & multi-player support	*Functional*	*Allows multiple teams and players to join and play simultaneously.*
Four timed rounds	*Functional*	*Each game consists of four rounds with specific time limits.*
Image rendering & guessing	*Functional*	*System renders images and will accept team guesses.*
Web accessibility	*Functional*	*Accessible through web browser platforms*
Scalability	*Non-Functional*	*Support growth in teams and players*
Performance	*Non-Functional*	*Real-time responses with minimal latency.*
Security	*Non-Functional*	*Restricts unauthorized access; securing user and score data.*
Maintainability & Modularity	*Non-Functional*	*Object-oriented design that supports updates, debugging, and enhancements.*
Reliability/Availability	*Non-Functional*	*System remains stable during game sessions.*

Design Constraints

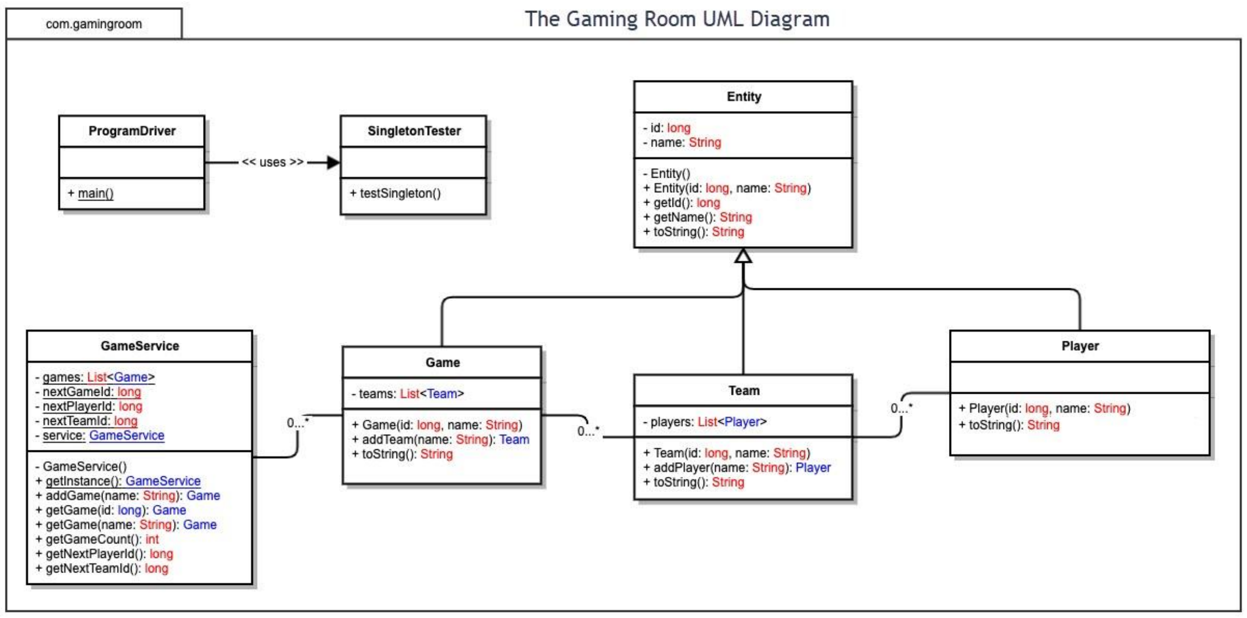
- **Language:** Developed in Java to ensure compliance with design principles.
- **Platform:** Web-based, ran in modern browsers across multiple operating systems (Edge, Chrome, Firefox, etc)
- **Game Rules:** Only one game instance will exist in memory at a time; all team and player names must be unique (No duplicates)
- **Performance:** Must support timed rounds with responsive images and real-time guess validation.
- **Scalability:** Must allow multiple teams and players to play without performance degradation.
- **Security:** Restricts unauthorized access and protect user data.
- **Resources:** Limited by available memory, processing power, and network connectivity during concurrent sessions.
- **Standards:** Must follow object-oriented design practices and standard coding conventions to ensure maintainability.

System Architecture View

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

The UML diagram for *The Gaming Room* illustrates an object-oriented design that supports the client's requirements. The **Entity** superclass defines common attributes (**id** and **name**) shared by the **Game**, **Team**, and **Player** classes, which reduces redundancy. **GameService** manages all active games using the Singleton pattern that ensures only one instance of the game exists in memory at a time. Each **Game** contains multiple **Team** objects, and each **Team** contains multiple **Player** objects. This reflects the real-world structure of gameplay. With this domain hierarchy we can enforce unique names, support scalability, and maintain consistency across the web-based system.



Evaluation

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.

Development Requirements	**Mac**	**Linux**	**Windows**	**Mobile Devices**
Server Side	MacOS is stable and secure, but it isn't really the go-to choice for running servers because of its limitations. It is functional but the hardware costs are much higher, so it's not the most budget friendly.	Linux is the most popular median choice for web servers. It's free, flexible, and reliable but will take a little more technical education to set up and manage.	easy to work with and fit right into environments already using Microsoft tools. The downside is the licensing costs and frequent updates can sometimes get in the way.	don't act as servers, but originally how people accessed the game. The server needs to deliver smooth transition across different phones and tablets.
Client Side	work well for client use, but not everyone has one since they're more expensive. So, using it would limit the client reach.	Linux users are a smaller group, but the system runs lightweight browsers and open-source tools well, so it's good for developers and power users.	Windows has the biggest user base, so supporting it means reaching the widest audience. Most browsers and devices run easily on it.	Most users will utilize this platform so designing for both iOS and Android with different screen sizes is key, so the game runs well for all.
Development Tools	some programs are only supported on Apple hardware, which can get expensive.	very customizable, making it great for development, especially for servers.	Windows integrates well with Microsoft Azure for cloud deployment and offers strong debugging and testing environments.	Mobile development uses Android Studio for Android and Xcode for iOS. Cross-platform tools like React Native or Flutter help make one build work across different devices.

While Windows and Mac OS offer user-friendly interfaces and strong development tools, they fall short in scalability and cost-efficiency. Windows is widely used in enterprise environments but requires licensing fees and isn't as popular for web hosting. Mac OS is stable and secure but less flexible for server-side deployment.

Mobile platforms are essential for client-side access but aren't suitable for hosting the game backend. They rely on platform-specific tools like Android Studio and Xcode, which limit cross-platform deployment.

Linux, by contrast, offers the best balance of performance, security, and scalability, making it the ideal choice for The Gaming Room's expansion.

<u>Recommendations</u>

After evaluating Mac OS, Windows, Linux, and mobile platforms, Linux stands out as the best fit for The Gaming Room's expansion. It is an open source, eliminating licensing costs, and is widely used in scalable server environments. Linux supports distributed systems and virtual memory management; all of which are essential for hosting multiple game instances and handling high traffic. Linux also offers built-in firewalls, role-based access control, and tools for managing permissions. It also supports SSL/TLS encryption and regular patching, which helps protect user data and gameplay across platforms. Compared to Windows and Mac, Linux provides more flexibility and long-term value, especially for a growing company with performance and scalability in mind.

These recommendations directly support the client's functional and non-functional requirements, including scalability, security, and maintainability. By choosing Linux, the system can grow with user demand while maintaining performance and data protection.

1. **<u>Operating Platform</u>**: Linux is the recommended platform. It's widely used for hosting servers, free of licensing costs, and highly scalable. It requires more technical expertise but provides the most flexibility and long-term value for a growing online game.

2. **<u>Operating Systems Architectures</u>**: Linux provides a modular architecture where the kernel handles core tasks like process and memory management, while user processes run independently. This separation makes the system efficient, secure, and stable. Because Linux is open source, developers can customize the environment to meet the exact needs of the game while still maintaining cross-platform compatibility.

3. **<u>Storage Management</u>**: For storage management, Linux supports multiple file systems such as ext4, XFS, and Btrfs. Ext4 is a reliable choice because it balances speed and stability while supporting large files and volumes. These file systems allow fast data access, secure handling of player information, and flexible scaling as the game's user base grows.

4. **<u>Memory Management</u>**: Linux uses virtual memory management to ensure efficient use of system resources. It allocates memory dynamically, prioritizes active processes, and uses swap space when needed. For "Draw It or Lose It," this means smoother performance during peak usage, as the system can handle multiple teams and players without crashing or lagging.

5. **<u>Distributed Systems and Networks</u>**: To support communication across platforms, the application can use Linux-based distributed systems with load balancers and clustering. This setup allows the game server to scale horizontally, distributing traffic across multiple machines if needed. Network connectivity is maintained through standard protocols (TCP/IP, HTTP/HTTPS), and redundancy features reduce the risk of outages. If one server fails, others can take over, keeping the game online.

6. **<u>Security</u>**: Linux provides strong security features including user authentication, role-based permissions, and built-in firewall tools like iptables. For added protection, data can be encrypted in transit using SSL/TLS and stored securely in databases. Regular updates and patch management further reduce vulnerabilities. These protections ensure user data and gameplay remain safe across different platforms.

All in all, Linux gives you long-term flexibility and cuts out licensing costs, but it's not plug-and-play for everyone. Getting it set up right might take some technical knowledge and team training upfront. That said, the payoff is worth it as Linux's scalability and built-in security features make it a smart investment for growing a platform such as "Draw It or Lose It.

Tools like Docker and Kubernetes help keep things futuristic. These tools make it easier to roll out updates, manage services across environments, and stay adaptable as the game evolves and new tech comes into play.

Implementation Strategy

To fully transition to the multiplatform system the following phase implementation timeline will be executed:

1. **Environment Setup:** Deploy Linux-based servers with Java runtime environments and web server. This will support cross-platform compatibility and browser access.
2. **Code Refactoring:** Modularize the game logic to separate UI, mid-tier, and data storage. This allows every layer to scale independently and support its future updates.
3. **Security Integration:** Implement role-based access control, session isolation, and encrypted storage. Use a token-based authentication for secure user access.
4. **Agile Sprint Planning:** Development will be broken into sprints, focusing first on the core then expanding to browser compatibility and responsive design.
5. **Testing and Risk Mitigation:** Test across Chrome, Firefox, and Edge to ensure consistent performance. Use load testing tools to simulate multiple users and identify bottlenecks. Set up fallback systems to maintain uptime during peak usage.
6. **Deployment and Maintenance:** Roll out the platform in stages, starting with internal testing, followed by a public beta. Document the system thoroughly to support maintainability and future updates.

Conclusion

The Gaming Room's transition to a scalable, web-based platform requires a platform that is secure, flexible, and future-proof. After carefully considering our options, Linux will meet all the clients' requirements and is the best fit for the project. It's open-source, very reliable, and loaded with tools that support strong server performance and security.

Linux offers containerized services, modular architecture, and distributed systems that can handle multiple teams, keep gameplay smooth, and will stay online even during heavy traffic. Pairing these technical choices with a phased Agile rollout means the team can build smart, test thoroughly, and adjust as needed. This approach doesn't just meet the client's current goals, but also set them up for future success.