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	12/12/2025	Jamaal Mims	Application Evaluation

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 expanding its Android-only game *Draw It or Lose It* into a cross-platform web-based system. The new version will support multiple teams, multiple players, and unique naming validation across the entire application. The backend will be structured so only a single instance of the game-managing service exists at a time, ensuring data consistency across concurrent players.

To meet these needs, this design uses critical software design patterns—primarily the Singleton Pattern for centralized game state management and the Iterator Pattern for searching and validating unique names. This document outlines the constraints, domain model, platform evaluation, and final recommendations to create a scalable, maintainable, and secure web application.

Requirements

The *Draw It or Lose It* system must allow users to create and join game sessions, organize players into teams, and participate in turn-based gameplay that includes drawing, guessing, and score tracking. The system must enforce unique identifiers for games, teams, and players while supporting multiple concurrent users without data conflicts. A web-based interface must be provided so the game can be accessed from desktop and mobile browsers across different operating systems, and the system must persist necessary game data between sessions. From a nonfunctional perspective, the system must deliver acceptable performance and low latency during gameplay, scale to support an increasing number of users, and remain reliable in the presence of network interruptions or partial system failures. Security requirements include encrypting all client–server communications, protecting user data from unauthorized access, validating user input, and supporting secure authentication and session management. The system must also be maintainable and extensible, using modular and well-documented code to support future enhancements, monitoring, and troubleshooting.

Design Constraints

The design of the *Draw It or Lose It* system is constrained by the need for cross-platform compatibility, requiring the application to run on Windows, macOS, Linux, and mobile devices through a web-based interface without platform-specific installations. The server-side application must be developed using Java, which restricts the technology stack and necessitates execution within the Java Virtual Machine for memory and process management. The system must support multiple concurrent users while enforcing unique game, team, and player names, requiring careful synchronization and thread-safe design. Strong security requirements mandate encrypted communication, secure authentication, and protection of user data across all platforms. Because the system operates in a distributed environment, it is dependent on reliable network connectivity and must handle latency, outages, and reconnection gracefully. Cost considerations limit the use of proprietary software and favor open-source solutions, while maintainability and future

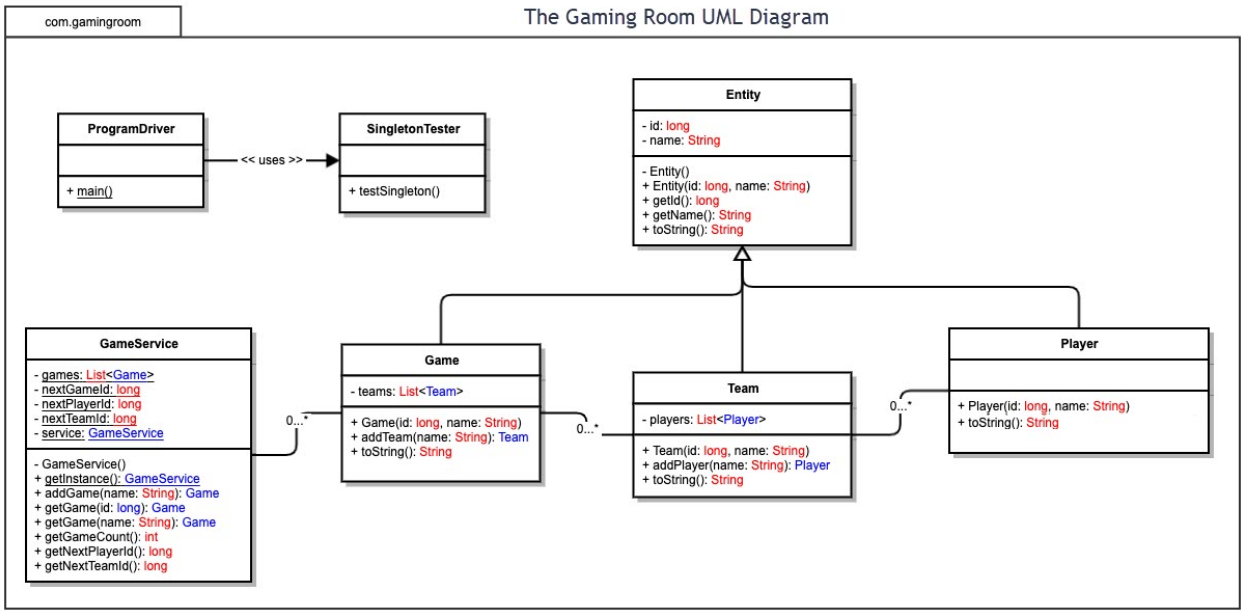
expansion constraints require a modular, scalable architecture that can evolve as the user base grows.

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 domain model for *Draw It or Lose It* centers on the relationships between the game session and the people participating in it. A Game represents a single active session and contains one or more Teams. Each Team represents a group competing in the game and contains one or more Players. A Player represents an individual user participating on a team. A central GameService class manages the creation and retrieval of games and enforces system rules such as uniqueness of game and team names. GameService is designed using the Singleton pattern so only one instance controls game management across the application, and it may use collection traversal (commonly supported with an Iterator) to safely manage and search through games, teams, and players. Together, these entities define how the system stores and organizes core game data and how gameplay sessions are managed consistently across multiple users.



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.

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	Reliable for development, not widely used for large-scale production hosting.	Industry standard for Java web applications; secure, scalable, lightweight, and cost-efficient.	Functional for hosting but introduces licensing costs and higher resource requirements.	Not viable for hosting due to hardware limits, connectivity, and battery dependence.
Client Side	Good for development and testing; smaller user market share.	Limited as a client environment, but excellent for compatibility testing.	Largest desktop user base; must support Chrome, Edge, and Firefox.	Essential for accessibility; requires responsive web design.
Development Tools	IntelliJ, Maven, JDK—excellent developer environment.	Best for back-end server deployment, supports Jenkins, Docker, Git.	Supports all major IDEs; good enterprise integration.	Requires responsive front-end; optional hybrid app frameworks.

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**: Based on the evaluation of macOS, Linux, Windows, and mobile platforms, Linux is the recommended operating platform for hosting the Draw It or Lose It server-side application. Linux provides industry-standard support for Java-based web applications, excellent performance under high concurrency, strong security controls, and superior scalability in both on-premises and cloud environments. It also avoids the licensing fees associated with Windows Server and the hardware cost and scalability limitations of macOS, making it the most cost-effective and flexible choice as The Gaming Room expands to a larger, web-based user base.

2. **Operating Systems Architectures**: The recommended architecture for Draw It or Lose It on Linux is a multi-tier, web-based architecture. The presentation layer runs in the user's browser (desktop or mobile) using HTML5, CSS, and JavaScript, ensuring portability across Windows, macOS, Linux desktops, and mobile devices. The application layer is a Java-based service (GameService) running in a JVM on the Linux server; this layer enforces business rules, coordinates game sessions, and implements the Singleton and Iterator patterns for unique game, team, and player management. The data layer is provided by a relational database (such as MySQL or PostgreSQL) running either on the same host or on a separate database server. This separation of concerns improves maintainability, allows independent scaling of tiers, and supports deployment in containerized or cloud environments.

3. **Storage Management**: For storage management, a relational SQL database such as MySQL or PostgreSQL is recommended. This data store will manage persistent information about games, teams, players, and their relationships, using primary keys for unique identifiers and foreign keys to maintain referential integrity between tables. This is a natural fit with the Entity, Game, Team, and Player classes from the domain model and supports constraints needed for unique names and IDs across the system. Transactions and row-level locking enable safe concurrent updates from multiple users, while indexed queries provide efficient lookup for validation and gameplay operations. Backups, replication, and clustering can be used to maintain availability and data durability as the user base grows.

4. **Memory Management**: The Draw It or Lose It application will run within the Java Virtual Machine (JVM) on the Linux server, leveraging the JVM's built-in memory management and garbage collection mechanisms. The Singleton GameService instance will reside in the heap and maintain in-memory structures for active games, teams, and players. As players join or leave games, objects are created and later become eligible for garbage

collection when no longer referenced. The JVM's generational garbage collector automatically reclaims unused memory, reducing the risk of memory leaks when combined with good coding practices (e.g., removing entities from collections when a game ends). Linux itself provides virtual memory management, paging, and process isolation to ensure that the JVM has sufficient memory while protecting the overall stability of the host system. If needed, heap size and garbage collection parameters can be tuned to optimize performance under high concurrency.

5. **Distributed Systems and Networks**: To support communication between various platforms (desktop and mobile browsers across different operating systems), the system will use a distributed, client–server model built around RESTful web services. The server exposes HTTP/HTTPS endpoints that the browser client calls to create games, join teams, submit answers, and update game state. Because REST is stateless, each request contains the necessary context (such as an authentication token and game ID), making the system more resilient and easier to scale horizontally behind a load balancer. Real-time features—such as live drawing updates or scoreboards—can be supported through techniques like WebSockets or periodic AJAX polling from the client to the server. The design must account for network variability and outages by implementing timeouts, retries, and graceful error handling on the client side. If connectivity is lost, the client should notify the user, attempt reconnection, and resynchronize game state from the server when the connection is restored.

6. **Security**: Security is critical for protecting user information and game integrity across all platforms. All communication between clients and the Linux server must occur over HTTPS using TLS/SSL to prevent eavesdropping and tampering. User credentials (if accounts are used) must be stored using strong, salted hashing algorithms rather than plain text. Authentication and session management can be handled with secure, time-limited tokens (such as JWTs) or established frameworks that implement OAuth2 or similar schemes. On the server, input validation and output encoding will protect against common web vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Role-based access control will restrict administrative operations, and least-privilege principles will be applied to database accounts and system services. Logging and monitoring on the Linux platform will help detect suspicious activity, while regular patching and configuration hardening will keep the operating system and application stack secure over time.