Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

GAME MASTER
CS 230 Project Software Design
Version 4.0

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

Table Of Contents

Document Revision History

Version	Date	Author	Comments
1.0	<07/21/25>	<Hamna Khalid>	Initial Draft Of Software Design
2.0	<08/07/25>	<Hamna Khalid>	Revised Executive Summary, Design Constraints, Domain Model, Evaluation, And Recommendations Per Instructor Feedback.
3.0	<08/07/25>	<Hamna Khalid>	Finalized Evaluation for Project Two
4.0	<08/17/25>	<Hamna Khalid>	Finalized Evaluation and Recommendations, corrected UML and citations for Projects One–Three.

Executive Summary

Creative Technology Solutions (CTS) has been contracted by *The Gaming Room* to expand their current Android-only game, ***Draw It or Lose It***, into a web-based, multi-platform solution. The new application must support multiple users and teams playing concurrently, with strict rules around name uniqueness and single-instance control in memory. CTS will serve as the lead technology consultant, responsible for designing the software architecture and advising on the development environment.

To proceed efficiently, CTS recommends a cross-platform Java-based solution, ensuring compatibility with diverse hardware and operating systems. This solution will implement key software design patterns, including Singleton (to restrict the game service to a single instance) and Iterator (to manage collections of games, teams, and players).

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

Moreover, the project demands careful consideration of the operating platform environment, not just the software. The underlying hardware must support scalable, distributed systems, with enough memory and processing power to render stock images smoothly, ensure user authentication across sessions, and maintain performance in real time.

Requirements

The client requires a lightweight application to manage a list of games, ensuring:

- Only one instance of the game service exists in memory (Singleton Pattern)
- All stored games can be easily iterated through and retrieved (Iterator Pattern)
- The application must be extensible and follow clean code standards
- The design should support future deployment in a distributed, web-based environment

Design Constraints

Developing ***Draw It or Lose It*** as a web-based distributed application requires addressing multiple operating platform constraints:

Scalability Across Environments

The application must support three or more environments: development, testing (UAT & QA), and production. These environments need infrastructure that can scale on demand. A cloud-based platform (e.g., AWS or Azure) is recommended to dynamically allocate resources based on traffic and development needs.

Memory and Storage Management

Stock images used for the game clues are high-resolution and must be served quickly to multiple clients. This requires the server to have sufficient RAM to render images smoothly and storage to house an extensive library.

Authentication and Security

The game is evolving from a single-player mobile app to a multi-user system requiring secure authentication (e.g., OAuth). Security measures must restrict simultaneous logins and protect user/team data.

Concurrency & Single-Instance Management

Only one instance of the game service can exist in memory at a time. This requires Singleton implementation and careful multi-threading strategies to prevent race conditions or state corruption.

Administrative Complexity

Future versions may include an admin interface to manage users, teams, and stats. This demands flexible architecture.

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*** was developed using the course template and incorporates the instructor's guidance to ensure accuracy. It centers on a singleton service that manages a collection of Game objects and two driver/test classes used to launch and verify the design. This correct model removes the earlier (incorrect) inheritance with Entity/Team/Player and aligns the code and SDD one-to-one.

Classes and responsibilities

GameService (Singleton)

- ***Purpose:***

  A single, shared coordinator that owns all game instances in memory.

- ***Attributes:***

games: List<Game> – in-memory list of games

nextGameId: long – auto-incrementing identifier source

service: GameService – private static reference to the sole instance

- ***Operations:***

getInstance(): GameService – returns the single instance

addGame(name: String): Game – creates and registers a new game with a unique ID

getGame(index: int): Game – returns a game by list index

getGame(id: long): Game – returns a game by unique ID

getGame(name: String): Game – returns a game by name

getGameCount(): int – returns number of games

Game

- ***Purpose:***

Lightweight domain object representing one game.

- ***Attributes:***

id: long – unique identifier

name: String – immutable, human-readable name

- ***Operations:***

getId(): long

getName(): String

toString(): String

ProgramDriver

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

- ***Purpose:***

  Entry point; contains main() to bootstrap the application and demonstrate core behaviors.

SingletonTester

- ***Purpose:***

  Verifies that GameService enforces a single instance and that lookups/creation behave as expected via testSingleton().

Key relationships and constraints

- **Aggregation:** GameService aggregates 0..* Game objects (GameService → Game).
- **Uniqueness:** GameService.addGame assigns IDs from nextGameId and should enforce unique game names.
- **Encapsulation:** All fields are private; access is via getters only (no public setters on Game).
- **Iteration:** Clients iterate through games using standard Java iterators, supporting simple reporting and validation.
- **Single instance guarantee:** Only GameService.getInstance() can create/access the service; constructors are not publicly exposed.

Rationale

This model is minimal, testable, and scalable: GameService centralizes coordination, Game remains immutable/simple, and the driver/test classes separate execution from verification. It matches the template UML and ensures the code, UML, and SDD stay in sync.

UML Diagram

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

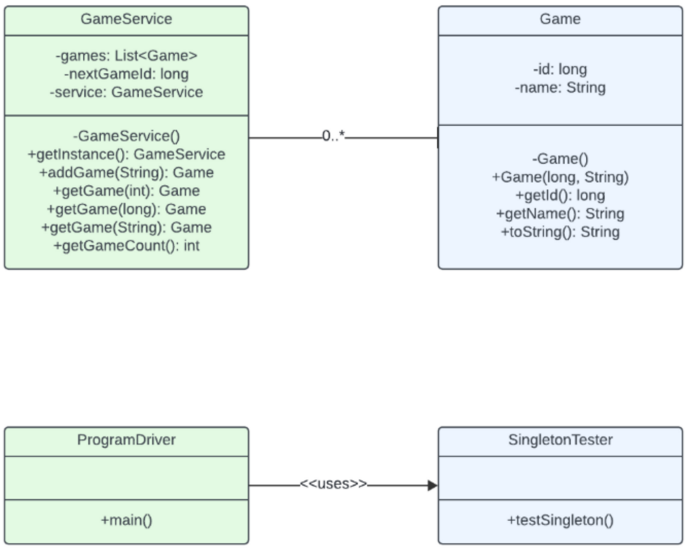

Figure 1: UML Class Diagram for "Draw It or Lose It"

Evaluation

Each operating platform was carefully evaluated to determine its ability to support the expansion of ***Draw It or Lose It*** into a scalable, distributed application. The comparison considers three major areas: server deployment capabilities, client access through browsers and operating systems, and development tool support.

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Provides a stable UNIX-based environment but is less common for enterprise-scale servers; requires licensing.	Open-source and UNIX-based, Linux offers scalability, security, and low cost, with strong enterprise adoption and support for virtualization, containers, and cloud services (Linux Foundation, n.d.; Tanenbaum & Bos, 2015).	Popular in enterprises, with seamless integration into Microsoft ecosystems. Supports virtualization and cloud services, but licensing and resource costs are higher (Microsoft, n.d.).	Not practical for servers due to hardware limitations, connectivity issues, and lack of enterprise-grade support.
Client Side	Safari dominates on macOS and iOS; Chrome and Firefox are also available, but less common among Apple users.	Primarily used by developers; supports Chrome and Firefox, but is not a mainstream client OS for everyday users.	Chrome and Edge dominate usage; Firefox is also popular. Windows provides broad compatibility and remains the most widely adopted desktop client OS.	Access is driven by mobile browsers: Chrome for Android and Safari for iOS. Applications must be optimized for responsive mobile browsing (StatCounter, 2023).
Development Tools	Xcode is required for iOS apps; Eclipse and IntelliJ IDEA support Java development on macOS.	Excellent development environment with Eclipse, IntelliJ IDEA, Docker, and Kubernetes. Well-suited for scalable backend systems.	Visual Studio, Eclipse, and IntelliJ IDEA offer strong support for both GUI and backend development. Common choice in enterprise settings.	Android Studio (Java/Kotlin) and Xcode (Swift) are required for native apps. Flutter and React Native streamline cross-platform development (Google Developers, n.d.; React Native, n.d.).

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

From this evaluation, Linux emerges as the most effective server platform due to its open-source flexibility, scalability, and strong support for modern development practices such as virtualization and containerization. On the client side, broad browser compatibility is essential, with Chrome and Safari leading across desktop and mobile environments. Development should focus on using established IDEs and cross-platform frameworks to streamline deployment to multiple devices. Together, these findings ensure that the recommended environment balances cost, performance, and user accessibility.

Recommendations

Based on the evaluation of operating platforms, client requirements, and development considerations, the following recommendations provide a comprehensive plan for deploying ***Draw It or Lose It*** as a scalable, secure, and cross-platform application. These recommendations address the server environment, operating system architecture, storage and memory management, distributed networking, and security to ensure that the game performs reliably across multiple platforms.

Operating Platform

Linux (Ubuntu Server or Red Hat Enterprise Linux) is recommended as the primary server platform for ***Draw It or Lose It***. Linux is open-source, cost-effective, highly scalable, and widely used in enterprise environments. It offers stability, strong community support, and seamless integration with cloud services, making it ideal for a multi-user, distributed application (Tanenbaum & Bos, 2015).

Operating System Architectures

Linux's modular UNIX-based architecture supports multitasking, efficient process scheduling, and robust memory protection. Its monolithic kernel is optimized for high performance and reliability, which is essential for managing concurrent users in a gaming environment. Additionally, Linux is fully compatible with containerization technologies such as Docker and Kubernetes, enabling consistent deployment across diverse computing environments (Linux Foundation, n.d.).

Storage Management

For storage, a hybrid approach is recommended:

- **Cloud-based distributed storage** (e.g., AWS S3 or Azure Blob Storage) for scalability, redundancy, and global access to game assets and user data.

- **Linux-native file systems** such as ext4 or XFS for local server storage, providing reliability and efficient file handling.
  This combination ensures that assets, images, and game session data remain secure, scalable, and easily retrievable (Amazon Web Services, 2023).

Memory Management

Linux implements advanced memory management techniques, including virtual memory, paging, and demand paging, to optimize performance under heavy load. It also employs security mechanisms such as Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP), which protect against memory-based attacks (Tanenbaum & Bos, 2015). These features ensure the system can efficiently handle simultaneous gameplay, resource-intensive media files, and user interactions without degrading performance.

Distributed Systems and Networks

Draw It or Lose It should be deployed in a distributed client-server architecture. Key recommendations include:

- Using **RESTful APIs** over HTTPS for cross-platform communication.
- Supporting **real-time gameplay** with WebSockets.
- Employing **container orchestration** (Docker and Kubernetes) for automated scaling and resilience.
- Implementing **load balancers** and redundant servers to reduce downtime during outages.

This distributed approach ensures scalability, high availability, and seamless cross-platform communication.

Security

Protecting user data and gameplay information is a top priority. Security recommendations include:

- **TLS/SSL encryption** for secure data transmission.
- **OAuth 2.0 and JWT tokens** for authentication and session management (Oracle, n.d.).

- **Role-Based Access Control (RBAC)** to restrict admin privileges and enforce the principle of least privilege.
- **Encryption at rest** for stored user data to prevent unauthorized access.

These security measures align with best practices and industry standards to maintain data integrity, confidentiality, and user trust.

By adopting Linux as the server platform, leveraging containerization and cloud-based storage, and implementing advanced memory management and distributed networking, ***Draw It or Lose It*** will be well-positioned for long-term scalability and reliability. The inclusion of robust security measures ensures that user data remains protected, while a cross-platform development approach guarantees accessibility for both desktop and mobile users. Collectively, these recommendations align the system architecture with the client's needs and industry best practices, providing a strong foundation for future growth and enhancements.

References And Citations

- Amazon Web Services. (2023). *What is cloud computing?* https://aws.amazon.com/what-is-cloud-computing/
- Apple. (n.d.). *Xcode overview*. Apple Developer. https://developer.apple.com/xcode/
- Flutter. (n.d.). *Build apps for any screen*. https://flutter.dev/
- Google Developers. (n.d.). *Android Studio*. https://developer.android.com/studio
- Linux Foundation. (n.d.). *About Linux*. https://www.linuxfoundation.org/about/
- Martin, R. C. (2022). *Clean architecture: A craftsman's guide to software structure and design*. Prentice Hall.
- Microsoft. (n.d.). *Windows Server documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/windows-server/
- Oracle. (n.d.). *The Java tutorials*. Oracle. https://docs.oracle.com/javase/tutorial/

Hamna Khalid
CS-230
Southern New Hampshire University
August 17, 2025

- Oracle. (n.d.). *OAuth 2.0 and JSON web tokens*. Oracle. https://www.oracle.com/security/
- React Native. (n.d.). *Learn once, write anywhere*. https://reactnative.dev/
- StatCounter. (2023). *Desktop vs mobile vs tablet market share*. https://gs.statcounter.com/platform-market-share/
- Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.