Draw It or Lose It Web App

CS 230 Project Software Design Template

Version 1.1

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.1	06/08/2025	Eric May	Added evaluation and recommendations

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 transitioning their Android-based game, Draw It or Lose It, into a web-based, multi-platform application. This requires a scalable and secure software architecture capable of handling multiple teams, players, and game instances with strict uniqueness requirements. The proposed design uses Java and employs design patterns like Singleton and Iterator to manage memory, enforce unique naming, and maintain structure. This document outlines the architectural decisions, object model, platform evaluations, and implementation strategy to ensure a seamless and maintainable development process. As part of this process, the document includes a detailed evaluation of Linux, macOS, Windows, and mobile platforms to guide development strategy for both server hosting and cross-platform client support.

Requirements

- Each game must support one or more teams.
- Each team must support multiple players.
- Game, team, and player names must be unique.
- Only one instance of the GameService class (the controller) can exist at runtime.
- The game must be easily portable to web and mobile platforms.
- System must be designed using object-oriented principles.

Design Constraints

1. Singleton Limitation: To meet the requirement that only one instance of the game exists in memory, the design uses the Singleton pattern for the GameService class.
2. Unique Names: Game, team, and player names must be unique. To enforce this, the system iterates through existing entries before adding new ones.
3. Platform Portability: Application logic must be cross-platform and modular. Java was chosen for its portability and scalability.
4. Web Compatibility: Code must be compatible with RESTful APIs and future front-end frameworks.
5. Scalability: As multiple users and games may interact concurrently, the structure must support expansion while maintaining performance.

System Architecture View

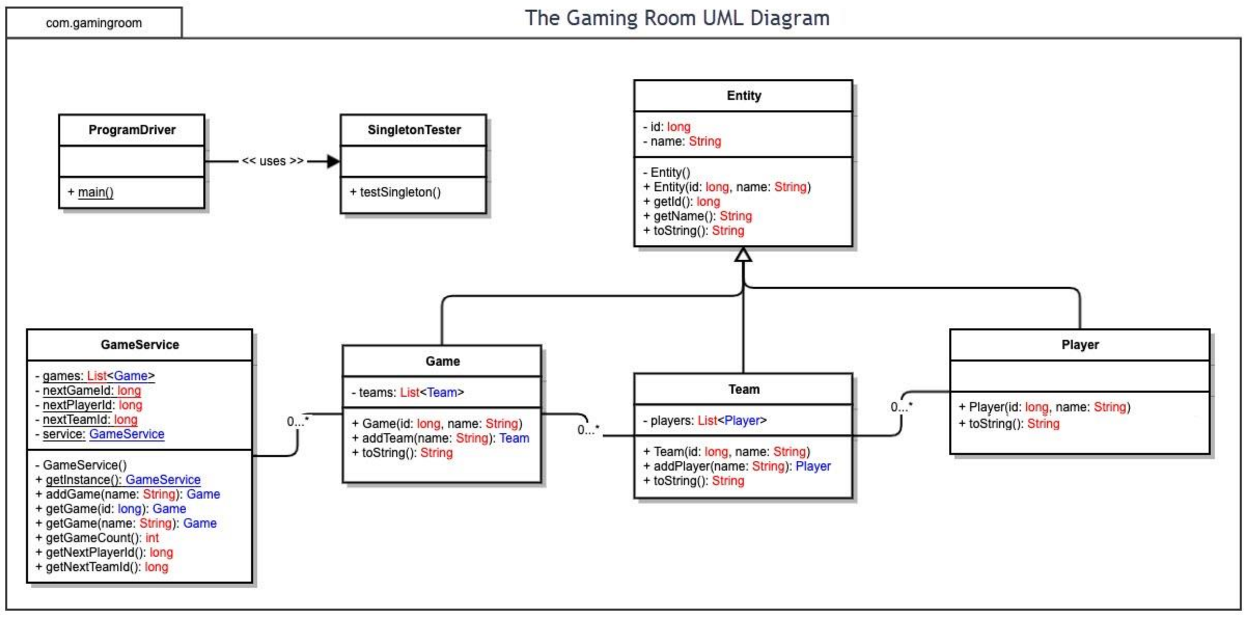
Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

Domain Model

The UML diagram defines an object-oriented system with the following relationships:
- Entity is the base class for Game, Team, and Player, encapsulating common attributes (id, name).
- Game holds multiple Team objects.
- Team holds multiple Player objects.
- GameService is a Singleton that manages all instances of games, teams, and players.

OOP Principles used:
- Inheritance: Reuse and generalization through Entity.
- Encapsulation: Controlled access to class attributes.
- Abstraction: Hides shared details in the base class.
- Design Patterns:
  - Singleton (GameService)
  - Iterator (to enforce uniqueness of names)

This structure ensures modularity, code reuse, and easy maintenance.



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	Good for local development and testing, limited in production use due to macOS hardware requirements	Excellent for web hosting, open-source, scalable, secure, and widely supported	Supported through Windows Server, reliable but has higher licensing and infrastructure costs	Not suitable for backend server logic or hosting
Licensing Cost	Requires Mac hardware, macOS Server is no longer actively supported	Free and open-source, no licensing costs for OS	Windows Server licenses can be costly, also may require CAL (Client Access Licenses)	Not Applicable
Client Side	Supports Safari and Chrome, less commonly used for gaming clients	Chrome and Firefox widely supported, compatible with HTML5 apps	Most commonly used desktop OS, compatible with all major browsers	Requires responsive web design or native mobile apps, Safari and Chrome must be tested separately
Development Tools	IntelliJ IDEA, Eclipse, and Xcode, limited to Mac hardware	Strong open-source tooling, supports CI/CD pipelines	Enterprise tools like Visual Studio widely used, supports .NET/Java	Android Studio, Xcode, Flutter, and React Native for native or hybrid app development
Development Complexity	High for iOS-specific builds, cross-browser testing needed	Moderate, ideal for backend developers familiar with open-source tools	Moderate, commonly used, but may require separate Windows only configurations	High, requires different toolchains per platform unless using cross-platform frameworks
Scalability	Not optimized for scaling server applications	Highly scalable, cloud friendly	Scalable in enterprise environments, but resource intensive	Not applicable on server side, on client side it depends on app design

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**: We recommend using Ubuntu Server LTS, a Linux-based operating system, for backend hosting. It provides strong community support, high performance, and compatibility with containerization tools like Docker. This platform is ideal for supporting web and mobile clients connecting to the game service over the internet.

2. **Operating Systems Architectures**: Ubuntu Server utilizes a monolithic kernel architecture that allows all system services (e.g., process and memory management, file systems) to run in a single address space. This improves efficiency while retaining modularity through loadable kernel modules. For Draw It or Lose It, this architecture ensures quick access to system resources while maintaining stable performance during concurrent game sessions.

3. **Storage Management**: We recommend implementing the ext4 file system in combination with Logical Volume Manager (LVM) for local storage, allowing flexibility and data redundancy. For game-specific data like user preferences, saved progress, and leaderboard stats, we advise integrating a relational database management system such as PostgreSQL to handle structured data efficiently.

4. **Memory Management**: Ubuntu Server supports advanced memory management using virtual memory, paging, and swapping. These mechanisms allow the game to scale under heavy load by optimizing memory allocation. The kerel's buddy system for physical memory and application-level caching will ensure low latency and smooth performance during gameplay.

5. **Distributed Systems and Networks**: To support communication between clients on various platforms (Android, iOS, web), we recommend a RESTful API architecture hosted on the server. These APIs can be consumed by different client types using secure HTTPS protocols. Using Docker containers and Kubernetes orchestration, the server application can scale and recover from outages with features like load balancing, health checks, and redundancy. Inter-component communication will rely on secure and persistent network connections.

6. **Security**: Security is essential when storing and transmitting user data. We recommend implementing TLS/SSL encryption for all data in transit, and data-at-rest encryption using Linux-based file encryption or PostgreSQL-native options. For authentication, OAuth 2.0 or JWT (JSON Web Token) can manage secure user access. Ubuntu Server's built-in firewall and support for tools like AppArmor add another layer of protection at the operating system level.