

Game Room Design

**CS 230 Project Software Design Template**

Version 2.0

**Table of Contents**

**Document Revision History**

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 06-23-24 | Kain Mason | The software design revisions in this version include expanding platform support, detailing evaluations and requirements, clarifying design constraints,and providing tailored recommendations for the Gaming Room's application expansion. |

**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 a client of Creative Technology Solutions. It has a developed game application, called Draw It or Lose It, which at the moment runs only on Android systems. The client is now in need of expanding the game also on other platforms like Linux, Mac, Windows, iOS, and Android through a web-based distributed environment. This paper details every feature available for each, the advantages, weaknesses, and how the developers find it useful to help in decision-making by The Gaming Room in regard to its application expansion.

**Requirements**

The basic needs for Gaming Room are all about enlarging the game application to support several platforms, being compatible with many operating systems, and providing an HTML interface that is modern and responsive for the client on desktop and mobile. To do this, the server-side application will need to cope with scalability toward thousands of players.

**Design Constraints**

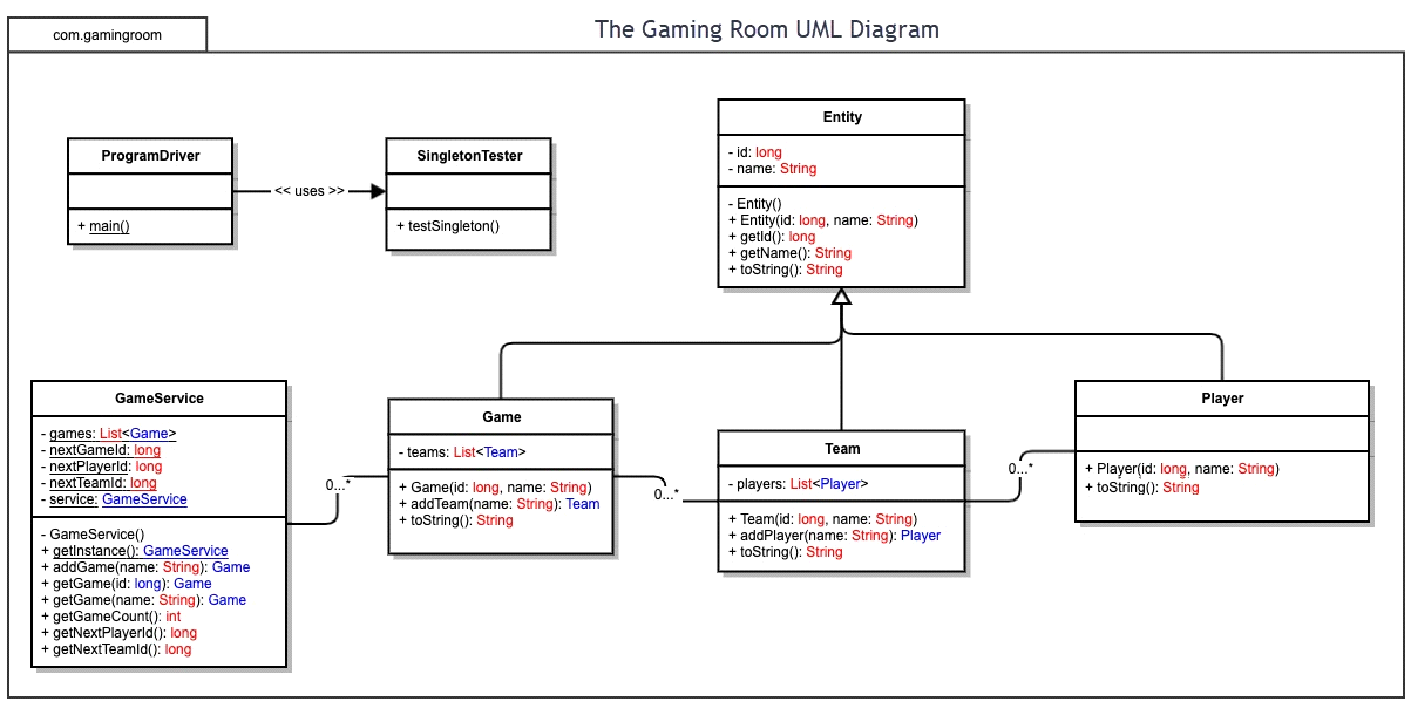
The game application has to be developed in a web-based distributed environment; hence, it has to be scalable for optimal performance and security. It should furthermore be platform-independent and device-agnostic. It has to be easy to develop at a low cost, with as little effort as possible, while requiring as little expertise as possible toward a full integration and functioning.

**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 following explanation is the class diagram of how The Gaming Room application design describes what classes we have, their methods, and their relationships. Program driver: This program driver uses the service GameService and SingletonTester to test the singleton pattern in GameService. Superclass Entity has attributes id and name based on that, we have Game, Team, and Player. For example, a Game may have numerous Teams, and then a Team may have multiple Players. The GameService uses the Singleton Design Pattern to manage the players, teams, and games within the application. At the core of the game are the object-oriented principles: inheritance, encapsulation, composition, singleton pattern, and polymorphism, all oriented toward ensuring the modularity, reusability, and maintainability of software.



**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** | Offers a user-friendly and reliable environment with robust security, but a smaller user base. macOS server supports server-based deployments with moderate potential licensing costs. | Highly stable and scalable, making it ideal for servers. It's cost-effective and open-source, though it may have a steep learning curve for developers. | Provides wide compatibility and is user-friendly. It supports server-based deployments with moderate to high licensing costs. There may be potential security concerns, but Windows servers can efficiently host web-based applications with additional security measures. | Typically use app stores for deployment, with moderate licensing costs. Their backends can be hosted on servers, which need to manage app data and user interactions efficiently. |
| **Client Side** | |  | | --- |   Ensure compatibility with web browsers. Development costs and time are moderate, with a need for expertise in macOS development. Seamless integration with other platforms is essential. | Ensure compatibility with web browsers. Development costs and time are lower due to open-source tools, but expertise in Linux development is needed. Seamless integration with other platforms is essential. | Ensure compatibility with web browsers. Development costs and time are moderate, with expertise in Windows development necessary. Seamless integration with other platforms is essential | Ensure compatibility with iOS and Android platforms. Development costs, time, and expertise vary based on fragmentation and platform-specific requirements. Seamless integration with other platforms is essential. |
| **Development Tools** | Use programming languages like JavaScript, Python, and Ruby. Common IDEs include Xcode, Visual Studio Code, and Sublime Text. Some tools may have licensing costs. | Use programming languages like JavaScript, Python, and PHP. Common IDEs include Visual Studio Code, Sublime Text, and PyCharm. Most tools are open-source and free. | Use programming languages like JavaScript, Python, and .NET languages like C#. Common IDEs include Visual Studio and Visual Studio Code. Some tools may have licensing costs. | Android development uses Java or Kotlin with Android Studio as the main IDE. iOS development uses HTML/CSS/JavaScript and frameworks like React or Angular with Xcode as the main IDE. Licensing costs for development environments are minimal. |

**Recommendations**

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

**Operating Platform**:I recommend deploying the application on Linux for server-side operations due to its stability, scalability, and cost-effectiveness. For client-side operations, the application should support Mac, Windows, iOS, and Android to maximize user reach and ensure compatibility across various devices.

**Operating Systems Architectures**: The Linux server architecture includes a robust kernel managing hardware interactions and system processes, a user mode layer isolating applications from the kernel for enhanced security, and flexible file systems like ext4, XFS, and Btrfs for efficient data management. Utilize a responsive HTML interface for Mac, Windows, iOS, and Android clients to ensure seamless user experience across platforms..

**Storage Management**: For storage management, leverage cloud-based solutions compatible with Linux such as AWS or Google Cloud Storage. These solutions offer scalability, high availability, data redundancy, and accessibility from any location, ensuring efficient data management and operational efficiency.

**Memory Management**: Linux employs various memory management techniques to ensure efficient memory usage and application performance. These include paging for swapping data between RAM and disk storage, memory mapping for faster data access, and virtual memory to extend available memory resources. These techniques are crucial for handling the demands of a web-based game application with numerous simultaneous users.

**Distributed Systems and Networks**: To enable Draw It or Lose It to communicate between various platforms, utilize RESTful APIs for efficient data exchange and interaction. Ensure robust network connectivity to minimize latency and enhance user experience, with redundancy and failover mechanisms to maintain system availability during outages. Manage dependencies effectively to ensure consistent performance and reliability across distributed systems.

**Security**: Implement comprehensive security measures to protect user information across all platforms. These measures include strong authentication mechanisms and access controls, data encryption both at rest and in transit, deployment of firewalls and Intrusion Detection Systems (IDS), and regular security audits and vulnerability assessments. Ensure compliance with security standards and best practices for each operating system to maintain robust security.