The Gaming Room

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/16/2024	Haley Candia Perez	The revisions clarify The Gaming Room's requirements and propose a scalable, secure web-based solution using a Linux backend and cross-platform tools. Updates include a detailed executive summary, identified design constraints, and refined recommendations to align with the client's needs.
1.1	12/01/2024	Haley Candia Perez	This revision evaluated Linux, Mac, Windows, and mobile platforms for hosting and supporting the Draw It or Lose It game, focusing on server capabilities, client-side development, and relevant tools. The findings outline each platform's advantages, limitations, and costs, guiding The Gaming Room's expansion to multiple platforms.

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 seeks to expand its existing Android-based game, Draw It or Lose It, into a web-based application that supports multiple platforms. This expansion aims to provide broader accessibility and scalability while maintaining the game's core mechanics. Our proposed solution includes a distributed web-based system that ensures compatibility across devices and platforms, leveraging modern development practices to manage unique identifiers, enable seamless team and player interactions, and deliver a consistent user experience. By addressing the outlined software requirements, we will create a scalable, secure, and user-friendly application to meet the client's goals.

Requirements

The business requirements are to develop a web-based version of the game that supports multiple platforms. Also, to ensure unique names for games and teams to enhance user experience and reduce conflicts. implementation of a single-instance memory system is also a requirement to ensure consistency. The technical requirements for this project include enabling multiplayer gameplay with multiple teams and players per team. Also, to ensure data integrity using unique identifiers for games, teams, and players. Finally, supporting scalability and reliability for concurrent users in a distributed environment.

Design Constraints

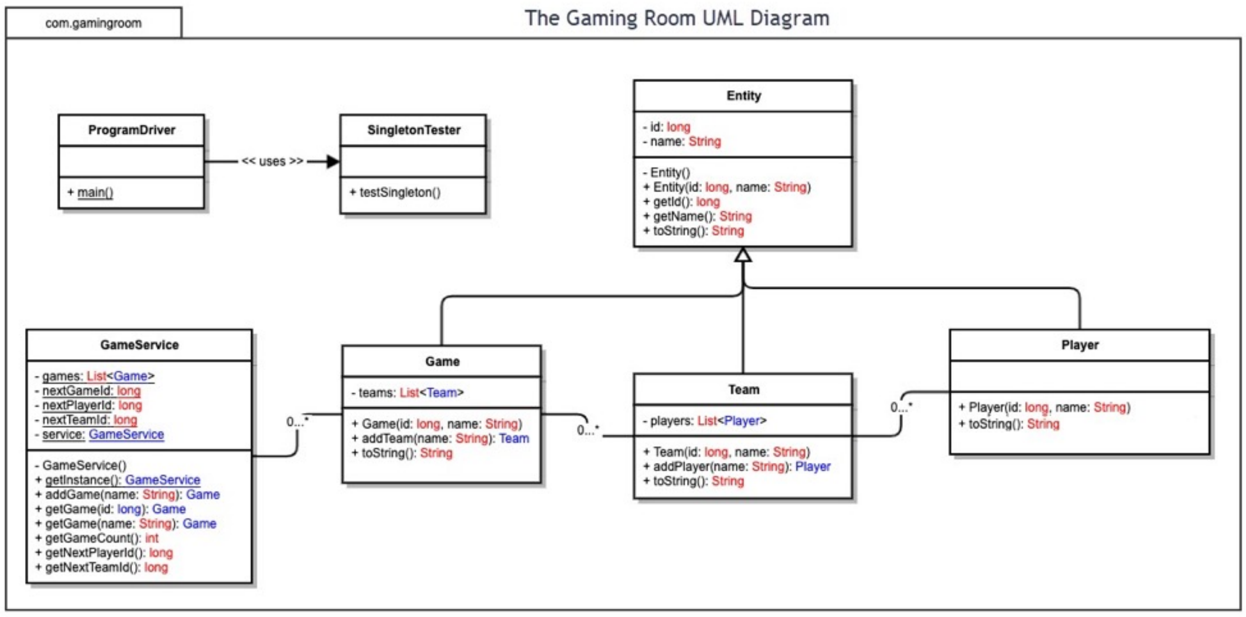
Some design constraints include implementing unique identifiers for games, teams, and players ensures data consistency and avoids conflicts. This will require a robust backend system with real-time validation and database indexing. Additionally, to maintain consistency, only one active game instance must exist at any time. Implementing a singleton design pattern or a centralized memory management system will ensure this. The game must also operate efficiently on various devices and platforms, requiring scalable backend architecture (e.g., cloud-hosted servers) and cross-platform development frameworks. Finally, protecting user data across platforms requires robust encryption, secure communication protocols (e.g., HTTPS), and authentication mechanisms.

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 illustrates a class-based system for managing games, players, and teams. The Entity class serves as a base class for Game, Team, and Player, promoting code reuse. Relationships like inheritance, association, and aggregation are used to structure the system. The GameService class manages games, while the Game, Team, and Player classes represent their respective entities. The diagram demonstrates the effective use of object-oriented principles such as inheritance, encapsulation, polymorphism, and abstraction to create a well-structured and maintainable software system.



<u>**Evaluation**</u>

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	Robust security and stability Seamless integration with Apple devices and software. Less common in enterprise environments Fewer hosting providers and limited support Higher hardware costs	Open-source, highly customizable, and cost-effective for hosting web-based applications. Offers strong performance and robust security. Requires expertise in command-line tools, potentially increasing the learning curve.	User-friendly GUI and excellent .NET and SQL Server support. Well-documented, accessible to teams with varying expertise levels. Higher licensing costs and hardware requirements compared to Linux.	Requires scalable cloud-based solutions like AWS, Google Cloud, or Azure. Must handle high traffic and ensure API efficiency for mobile-specific constraints. Challenges include managing bandwidth, latency, and user load.
Client Side	Strong security and stability Seamless Apple integration Less common in enterprise Fewer hosting options and limited support Higher hardware costs	Free and open-source nature reduces costs but demands expertise in Linux environments. Compatibility testing across multiple Linux distributions may add time to development. Developers need familiarity with Linux-specific tools like shell scripting.	Prevalence in businesses ensures faster development for experienced teams. Licensing fees increase costs but provide extensive software compatibility. Suitable for rapid development with robust tool availability.	Development needs separate builds for iOS and Android unless using cross-platform frameworks. Expertise in mobile UI/UX is essential to optimize user experience. Cost and time considerations for managing platform-specific constraints.
Development Tools	**IDEs:** Xcode, Apple's official IDE **Languages:** Swift, Objective-C **Tools:** Homebrew, Cocoa Frameworks **Licensing Costs:** Xcode is free, but Apple hardware is expensive.	**IDEs:** Visual Studio Code, Eclipse, JetBrains tools (e.g., IntelliJ IDEA). **Languages:** Python, Java, C++, JavaScript. **Tools:** Docker, Git, Vim, Bash, and package managers like apt or yum. **Licensing Costs:** Free or minimal for open-source tools.	**IDEs:** Visual Studio, Visual Studio Code. **Languages:** C#, Python, JavaScript, Java. **Tools:** PowerShell, WSL, SQL Server, and Microsoft Access. **Licensing Costs:** Visual Studio has tiered pricing plans based on features	**IDEs:** Android Studio (Android) and Xcode (iOS). **Frameworks:** Flutter, React Native, Xamarin for cross-platform development. **Languages:** Swift, Kotlin, Java, JavaScript. **Tools:** Firebase for backend integration and Postman for API testing. **Licensing Costs:** Flutter and React Native are free, costs for

<u>Recommendations</u>

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**: The recommended platform is a cloud-based distributed system using a Linux backend, with frontend support for Windows, macOS, and mobile platforms (iOS and Android). Linux offers excellent performance, flexibility, and cost-effectiveness, making it an ideal choice for developing, deploying, and scaling applications in distributed systems. Its robust ecosystem and compatibility with cloud services ensure seamless integration with various platforms.

2. **Operating Systems Architectures**:The system will leverage Linux servers to run the backend, ensuring stability and scalability through features such as efficient resource management, high availability, and robust security. The backend will support modular development, making it easier to maintain and scale individual components. Virtualization and containerization technologies will be used to ensure consistent deployment and streamlined operations across environments. For the frontend, cross-platform development approaches will ensure a consistent user experience across devices. A flexible framework for web development will be used to ensure compatibility with modern browsers and seamless integration with the backend. A cross-platform mobile development approach will enable a single codebase to support both iOS and Android platforms, reducing development time and ensuring feature parity across devices. This general approach ensures flexibility in tool selection while maintaining the ability to adapt to changing requirements and technologies. The integration between backend and frontend will utilize modern API standards for efficient communication and interoperability.

3. **Storage Management**: The recommended storage management system includes cloud-based storage solutions, combined with distributed file systems. These systems provide scalability, which offers dynamically adjustable storage to meet growing data needs. They also provide reliability, where redundant configurations ensure data availability and fault tolerance. Lastly, they provide global accessibility where cloud-based storage allows access from multiple regions, ensuring low latency for users worldwide. For database needs, integrating a relational database or a NoSQL database ensures efficient handling of both structured and semi-structured data, supporting the application's diverse requirements.

4. **Memory Management**: The Linux backend employs advanced memory management techniques to optimize the performance of Draw It or Lose It. Key techniques include demand paging, where the program loads only the necessary memory pages into RAM, conserving resources. It also includes virtual memory by Provides applications with a consistent address space, even when physical memory is limited. Caching accelerates read/write operations by storing frequently accessed data in memory. Finally, containerized resource management provides resource allocation across containers ensures optimal memory usage for all services.

5. **Distributed Systems and Networks**: To enable seamless communication between platforms, Draw It or Lose It will be implemented as a distributed application leveraging modular design principles and modern API-based communication. Key considerations will include Cloud-Native

Design which will use cloud platforms to deploy scalable services. Also, load balancing will help to distribute traffic across servers to maintain performance and prevent bottlenecks. Data Synchronization will ensure consistency between distributed components using appropriate synchronization tools. Utilizing cross-platform integration will allow frontend clients on various platforms to communicate with the backend over secure protocols. Finally, fault tolerance will provide cloud redundancy and failover systems minimize downtime during connectivity issues.

6. **Security**: Security is critical to protecting user information and maintaining trust. The following measures are recommended for implementation. Encryption will use TLS/SSL for secure communication between platforms. This will encrypt sensitive data stored in databases using strong encryption algorithms. Authentication and authorization will implement modern authentication protocols to manage user access securely. Also, regular updates will keep the operating system, libraries, and dependencies up-to-date to address vulnerabilities. Utilizing cloud security tools will be beneficial for access control and threat detection. Another measure that would prove useful would be deploying monitoring tools and configure alerts for suspicious activity. Lastly, leveraging platform-specific security features to enhance client-side security on mobile and desktop applications is recommended. By adopting these strategies, Draw It or Lose It can effectively expand to multiple platforms while ensuring performance, scalability, and security.