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

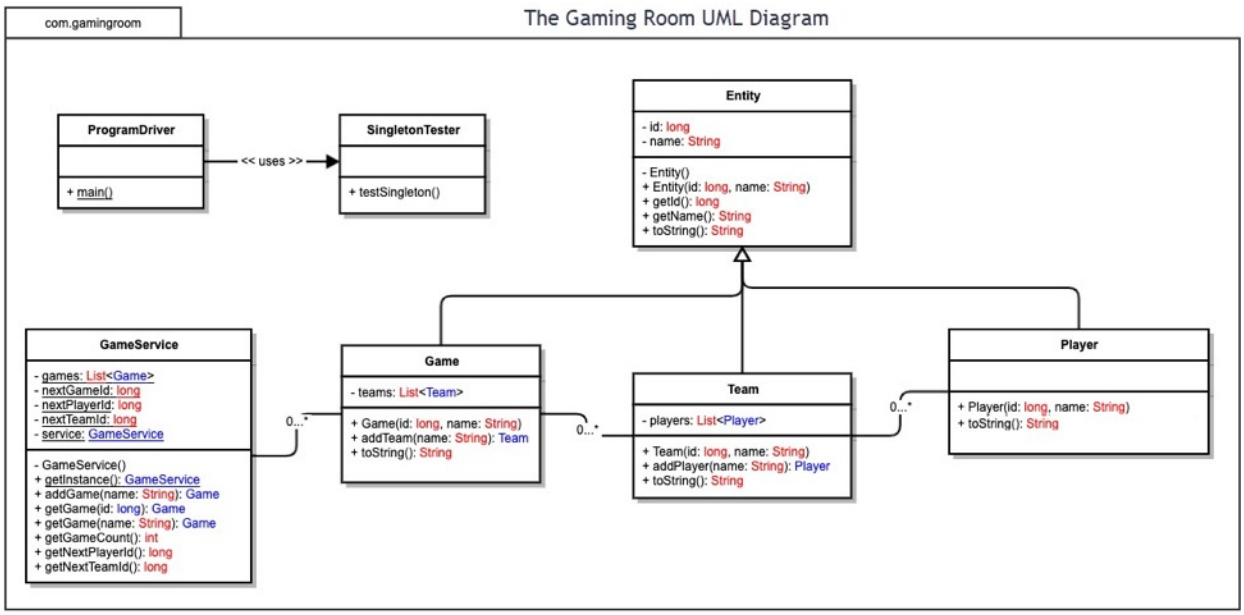
Some design constraints include implementing unique identifiers for games, teams, and players ensures data consistency and avoids conflicts. This will require a robust backend system with real-time validation and database indexing. Additionally, to maintain consistency, only one active game instance 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.



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	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

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**: The recommended platform is a cloud-based distributed system using a Linux backend for scalability and cost-effectiveness, with frontend support for Windows, macOS, and mobile platforms (iOS and Android).

2. **Operating Systems Architectures**: Linux servers will run the backend, providing stability and scalability, while the frontend will use cross-platform frameworks such as React.js for web compatibility and React Native for mobile.

3. **Storage Management**: A relational database such as MySQL or PostgreSQL is recommended for efficient data management, with cloud-based storage (e.g., AWS S3) for scalability and reliability.

4. **Memory Management**: The backend will employ efficient memory management techniques like caching (e.g., Redis) to support real-time gameplay and minimize latency. The singleton pattern will ensure only one instance of the game exists in memory at any time.

5. **Distributed Systems and Networks**: The application will use RESTful APIs for communication between frontend and backend, supported by a content delivery network to enhance performance. Dependencies such as database replication and load balancers will ensure resilience during outages.

6. **Security**: User information will be protected using secure encryption (e.g., AES) and communication via HTTPS. Multi-factor authentication and secure session management will provide additional layers of protection, ensuring compliance with data privacy standards across platforms.