

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 02/11/2024 | Donovan Hall | OS Evaluation |

**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 application design document describes the architecture and component interactions of a game management system. Central to this system is the GameService, a singleton class that provides global access and control over game sessions. Additionally, the document defines the Game, Team, and Player classes, which respectively manage game instances, group players, and handle player data. This document aims to offer a clear overview of the system's structure and its software component interactions.

# Requirements

The application's functional requirements include creating, retrieving, updating, and managing games, teams, and players. Non-functional requirements focus on performance, reliability, and scalability, particularly in regard to the singleton pattern of the GameService class. The application must also maintain a high level of data integrity and provide a responsive user experience.

# Design Constraints

Design constraints are centered around the singleton pattern used in the GameService class, which dictates that a single instance manages all game states. This design choice may impact the application's ability to scale and handle multiple concurrent requests.

# Rationale

The rationale for using a singleton pattern for the GameService class is to ensure a consistent point of access to game states, thereby preventing conflicts in game state management. Inheriting from a base Entity class for Game, Team, and Player streamlines object identification and management, simplifying database interactions and object life cycle management.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

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** | Advantages: Stable, reliable, integrates well with other Apple products.  Weaknesses: Higher costs due to hardware and software, less common for server use. | Advantages: Cost-effective (open-source, no licensing fees), customizable, wide community support, robust security features.  Weaknesses: Requires more technical expertise for setup and maintenance. | Advantages: Easy to use and manage, extensive support and documentation, integrates well with other Microsoft products.  Weaknesses: Licensing costs, potential security vulnerabilities. | N/A |
| **Client Side** | Advantages: Integration with other Apple products and services. High-quality graphic rendering and performance in web applications. Secure and stable environment for web browsing and application use.  Weaknesses: Limited customization options compared to Linux. Proprietary nature can limit certain applications or tools. | Advantages: High degree of customization and control over the browser environment. Open-source browsers and tools can be easily integrated. Strong community support for troubleshooting and enhancements.  Weaknesses: Compatibility issues with websites or web applications optimized for other operating systems or browsers. | Advantages: Broad compatibility with web standards and applications. Extensive support for various web browsers and tools. User-friendly interface for non-technical users.  Weaknesses: Potential security vulnerabilities requiring regular updates and patches. Some customizations and advanced features require additional software or settings adjustments. | Advantages: Responsive design and mobile-optimized applications provide a tailored user experience. Wide availability of apps and tools for enhanced functionality. Constant connectivity and integration with mobile services.  Weaknesses: Screen size and hardware limitations compared to desktop environments. |
| **Development Tools** | Advantages: Unified development environment with Xcode for developing applications across Apple's ecosystem. High performance and stability. Excellent support for graphic and multimedia applications.  Weaknesses: Restricted to Apple's ecosystem for certain development tools and applications. Higher cost of entry due to hardware and software expenses. | Advantages: Wide range of open-source development tools and libraries. Flexibility and control over the development environment. Strong support for server-side development and containerization technologies.  Weaknesses: May require more setup and configuration. The diversity of distributions can introduce compatibility challenges. | Advantages: Visual Studio provides a comprehensive development environment for .NET and other languages. Wide support for various development tools and languages. Large developer community and extensive documentation.  Weaknesses: Licensing costs for professional development tools. May require additional configuration for open-source tools and languages. | Advantages: Specialized IDEs like Xcode for iOS and Android Studio for Android offer integrated development environments tailored to mobile app development. Access to native APIs and hardware features. Strong ecosystems for app distribution (App Store for iOS, Google Play for Android).  Weaknesses: Platform-specific development require learning different programming languages and tools. |

## 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**: Linux OS stands out as a prime candidate for the server end of our gaming application. It’s an OS known for its stability and adaptability, particularly suitable for managing high-demand server tasks and supporting a vast array of development tools essential for game deployment.
2. **Operating Systems Architectures**: Client-side Considerations: We lean towards a Microkernel Design. It segments vital system processes into user-space operations, promoting a stable environment even if certain user space services fail.

Server-side Considerations: A Monolithic (yet modular) Kernel on the server side would be our backbone, offering the speed of in-kernel service execution without compromising on the flexibility thanks to its modular nature.

1. **Storage Management**: Efficient Data Handling: A Version Control File System would be advantageous, tracking alterations instead of overwriting data, thus optimizing storage space.

Memory Allocation: Introduce Dynamic Memory Allocation coupled with an automated cleanup system (garbage collection) to maintain an optimal memory footprint on client devices.

1. **Memory Management**: We recommend employing a dual strategy of Paging for memory space efficiency and Segmentation for orderly arrangement and protection of the game’s discrete functional segments. Items such as user interface, processing logic, and graphic rendering would be appropriate measures for memory management in this instance.
2. **Distributed Systems and Networks**: We recommend adopting a Peer-to-Peer Network Model for its robustness against single points of failure and its distributed nature which aids in load balancing during peak user engagement times.
3. **Security**: Authentication and Encryption: Incorporate reliable authentication flows, utilizing protocols like OAuth, and secure data transmission with TLS.

Ongoing Vigilance: Commit to Regular Software Updates and active monitoring to shield against emerging threats. Sandbox Environments will also be crucial, especially for the client-facing interfaces, to quarantine and manage executable processes securely.

# Server Side Evaluation

Linux offers a cost-effective, customizable solution with no licensing fees, but requires technical expertise. Mac provides stability and integration with Apple products but at higher costs. Windows is user-friendly, with extensive support but incurs licensing costs and potential security vulnerabilities. Mobile platforms are not traditionally used for hosting but could support cloud-based approaches.

# Client Side Considerations

A web-based application must be responsive and compatible across all platforms, utilizing HTML5, CSS3, and JavaScript. Responsive design ensures content layout adjusts across different screen sizes, with testing across browsers and devices for consistency.

# Development Tools

Development involves JavaScript for client-side, with Java, Python, or .NET for server-side. IDEs like Visual Studio Code, Eclipse, or Xcode are chosen based on the platform, with open-source tools reducing licensing costs. Multiple development environments may require additional training for the development team.