

Draw it or Lose It

# **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 | 09/22/2024 | Mario Zevallos | Initial design completion |
| 1.1 | 10/06/2024 | Mario Zevallos | Evaluation updated |
| 1.2 | 10/20/ | Mario Zevallos | Recommendation |

**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](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based version of their game, Draw It or Lose It, which currently exists only as an Android app. The web version should support multiple platforms and allow one or more teams to compete in each game, with unique names for both games and teams. Additionally, the system must enforce that only one instance of the game exists in memory at a time by using unique identifiers for each game, team, and player. The goal is to streamline the game's development to fit these requirements while ensuring scalability and smooth operation across platforms.

## [Design Constraints](#_2et92p0)

Platform compatibility: The game needs to work on multiple platforms, requiring cross-platform frameworks for web and mobile support.

Memory: Only one game instance can exist, so memory must be managed efficiently to prevent duplication.

Latency: Network communication needs to be optimized for real-time gameplay to minimize delays.

Scalability: The system must handle multiple teams and players at once, requiring scalable backend infrastructure.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram for The Gaming Room application outlines the relationships between classes using object-oriented principles. At the core is the Entity class, which provides shared attributes like id and name, as well as methods such as getId and getName. This class acts as a foundation for the Game, Team, and Player classes, which inherit these attributes and behaviors. Inheritance simplifies development by ensuring consistency across these entities and reducing the need to repeat code. The GameService class manages the creation and storage of game-related entities, including teams and players, while ensuring each entity has a unique identifier. The singleton pattern is employed in GameService to make sure only one instance of the class is ever created, centralizing game management and preventing multiple game instances from running simultaneously.

The Game class organizes teams, and in turn, the Team class manages players, following the composition principle, where smaller entities like teams and players are nested within larger ones, like games. Each team can contain multiple players, but players are tied to one specific team, maintaining clear associations and organization. The ProgramDriver class works with GameService to execute the game’s functionality, while SingletonTester verifies that the singleton pattern is functioning as intended. Additionally, encapsulation is used to protect data by keeping attributes private and providing access through public methods, ensuring secure and controlled manipulation of data. These object-oriented principles such as inheritance, composition, singleton, and encapsulation combine to create an efficient, organized structure that meets the software's requirements.

**"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** | Hosting a web-based application on macOS servers is possible, but it is not the most cost-efficient option. The macOS platform requires a $99 per year Apple Developer Program membership to access developer tools like Xcode and deploy macOS-based applications. While macOS offers a secure and stable environment for hosting web-based applications with strong integration into the Apple ecosystem, the associated costs can escalate due to the need for Mac hardware, which is more expensive than typical server setups using Linux or Windows. Additionally, macOS is less flexible compared to open-source alternatives, making it less popular for large-scale deployments where scalability and cost-efficiency are critical factors​ | Linux is the most cost-effective and preferred platform for hosting web-based applications. As an open-source operating system, it incurs no licensing fees, making it highly attractive for developers. Hosting services on Linux through providers like AWS come with minimal server costs, typically ranging from $5 to $40 per month for basic web hosting, depending on the scale of deployment. Linux is highly flexible and customizable, offering strong security and scalability, which makes it ideal for large-scale web applications. However, it requires a deeper level of technical expertise for proper configuration and maintenance, especially for optimizing performance and managing security​ | Windows Server licensing is more expensive, with costs starting at approximately $20 per month for entry-level hosting through cloud providers like Azure and increasing based on server configurations. Windows Server itself requires additional licensing, often starting at around $500 for a standard version or available through subscription plans for cloud-based solutions. Widely used in business environments, Windows servers offer seamless integration with Microsoft products and a user-friendly interface, making them a strong choice for enterprise environments. However, while powerful, they can incur significant long-term costs compared to Linux and may lack some of the security advantages inherent to open-source systems like Linux | Mobile web development uses frameworks like React Native, allowing developers to write code once and deploy it across both Android and iOS. Specialized development may require Java or Kotlin for Android and Swift for iOS, with Android Studio and Xcode as platform-specific tools. Cross-platform frameworks reduce costs and development time, but careful optimization is necessary to ensure compatibility, performance, and user experience across various devices and operating systems​ |
| **Client Side** | For client-side development on macOS, developers require the Xcode IDE, which is free but necessitates a $99/year Apple Developer Program membership for deploying applications. Xcode supports both macOS and iOS development, making it essential for those targeting Apple products. Mac development benefits from a unified ecosystem, which simplifies testing and optimization across devices. However, it can be costly and time-consuming due to the specialized skills required, particularly in Swift and Objective-C, to effectively develop for both macOS and iOS​. | Linux offers high flexibility and control in web development, making it ideal for developers who require customization. It supports a wide range of popular web languages like Python and Java, and as an open-source platform, it keeps development costs low. However, developing on Linux may require additional expertise and time to ensure compatibility across different Linux distributions, as developers must be familiar with various system configurations to maintain consistent performance. | Windows is highly compatible with a wide range of client-side web development tools and browsers such as Chrome, Edge, and Firefox. Developers can utilize Visual Studio, which supports multiple programming languages, including C#, Java, and JavaScript, offering significant flexibility for client-side development. This compatibility allows for a smooth user experience on Windows desktops. While Windows development is generally cost-effective due to the availability of numerous tools, some costs may be incurred, particularly with Visual Studio licensing, and expertise across various technologies is required for diverse software projects​ | Mobile client development focuses on various operating systems and device types, requiring careful attention to compatibility. Cross-platform frameworks help reduce costs, though expertise in optimizing for performance and user experience is critical. |
| **Development Tools** | Xcode is the primary development environment for macOS and iOS applications, widely used for coding in languages like Python, Java, and C. While Xcode is free, developers must pay a $99/year Apple Developer Program fee for app deployment. Additionally, Apple charges a 30% commission on in-app purchases and subscriptions, which can impact future app monetization​ | Linux developers commonly use open-source tools such as Visual Studio Code, which is free to use and widely supported on Linux systems. There are no licensing fees for development on Linux, making it the most affordable option for development teams. Many programming languages, including Python, C, and Java, work well on Linux, further contributing to its flexibility and cost-effectiveness for development​. | Windows developers commonly use Visual Studio, which supports multiple programming languages like Python, C, Java, and JavaScript. The full professional version of Visual Studio starts at $45 per month, with enterprise-level licenses available at higher costs, adding to the overall expense of the development process. Visual Studio is a versatile environment with a wide range of tools, including the .NET framework and Windows SDK, making it a robust choice for Windows development​ | Mobile development supports common languages like Java, Kotlin for Android, and Swift for iOS. Android Studio is a popular IDE for Android, while Xcode is used for iOS development. |

## 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**: I would recommend Windows as the operating platform. It's widely used, supports multiple development tools, and offers compatibility for web and mobile applications, making it a solid choice for expanding Draw It or Lose It to various computing environments.
2. **Operating Systems Architectures**: The Windows OS architecture is layered design, consisting of a kernel at its core that manages essential system functions like processes, memory, and security. It includes a hardware abstraction layer or HAL that handles interactions with hardware devices, allowing the game to run on various hardware configurations. Applications, like Draw It or Lose It, interact with user-mode components, making the system efficient and responsive for real-time gameplay.
3. **Storage Management**: Use SQL Server for the Windows platform. This tool provides structured data management, scalability, and integration with cloud services, ensuring the game data is stored securely and can scale with the application.
4. **Memory Management**: Windows employs virtual memory to allocate and manage memory dynamically, preventing issues like memory leaks. Caching mechanisms like Windows Cache Extension can be used to store frequently accessed data, ensuring smooth gameplay and better performance.
5. **Distributed Systems and Networks**: To communicate across platforms, the game can use APIs or web services that connect the backend to different clients like web, mobile, and desktop. This ensures real-time updates for players and teams, regardless of the platform. To handle large traffic and avoid downtime, the system can rely on cloud services with features like load balancing. It’s also important to have backups in case of network failures to keep the game running smoothly
6. **Security**: Security is essential to protect user data across platforms. Encryption should be used for data sent between the game and servers, and sensitive information must be stored securely. Windows offers built-in security tools like firewalls and encryption that can help safeguard the game. Implementing role-based access ensures that only authorized users access specific features, and regular updates keep the system protected from new threats.