

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

[**Recommendations 9**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/25/2025 | Kayle Church | Adds requirements, design constraints, summary and additional pertinent information the project requires. |
| 1.1 | 06/08/2025 | Kayle Church | Updates recommendations and system architecture view. |
| 1.2 | 06/20/2025 | Kayle Church | Updated recommendations to explain the architectural needs for the different operating platforms, and how to implement this architecture based on the client’s needs. |

**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 client aims to transform their existing Android-only application, *Draw It or Lose It*, into a multi-platform, web-based game. The new version should retain the core functionality of the current app while expanding its accessibility through cross-platform development. To achieve this, the project will require initial support with environment setup and development workflow optimization.

Key functional requirements include:

* The use of a cross-platform framework to ensure broad device compatibility.
* Enforcement of unique game and team names to prevent conflicts.
* A constraint that only one instance of a given game should exist in memory at any time to manage resources effectively.
* Support for dynamic team configurations, allowing one or more teams per game and multiple players per team.

We recommend beginning with a technical review of the existing Android application to inform the architectural approach for the web-based version. Following this, we will guide the client through the setup of a scalable, maintainable development environment and define a robust structure for managing multiplayer logic and session control.

## Requirements

The client requires the game to support flexible team configurations, allowing one or more teams per game, with multiple players assigned to each team. To ensure consistent gameplay and resource management, only one active instance of a game should exist at any given time.

To support these constraints, each game instance, team, and player must be assigned a unique identifier. Additionally, to enhance the user experience and prevent naming conflicts, the system must provide real-time validation to confirm the uniqueness of game and team names during creation or selection.

## [Design Constraints](#_2et92p0)

A primary constraint for this project is the need to develop the game for multiple platforms within a web-based distributed environment. Without team members experienced in cross-platform development, there is a risk of fragmented workflows, potentially requiring separate teams to develop and maintain platform-specific versions of the application. This can lead to increased development time, duplicated efforts, and higher maintenance overhead.

Another critical constraint is the requirement to ensure consistent functionality and performance across all platforms, fully meeting the client’s specifications. This adds complexity to the design and testing processes, as features must be implemented and validated uniformly across environments to maintain a seamless user experience.

To mitigate these challenges, adopting a robust cross-platform framework will be essential to unify development and streamline deployment across platforms.

## [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)

The UML class diagram illustrates the structure of *The Gaming Room* application and demonstrates key object-oriented programming (OOP) principles, including inheritance, encapsulation, and the singleton pattern.

At the core of the design is the Entity class, which acts as a superclass for Game, Team, and Player. These three classes inherit from Entity, thus sharing common attributes (id and name) and methods (getId(), getName(), and toString()). This use of **inheritance** promotes code reusability and simplifies management of shared behavior across these classes.

The GameService class plays a critical role in managing the overall game state. It maintains collections of games and tracks the next available IDs for games, teams, and players. The use of **encapsulation** is evident as these fields are kept private and accessed through public methods. Additionally, GameService implements the **singleton pattern**, ensuring only one instance of the service exists, which aligns with the client requirement of allowing only one active game instance in memory at a time. This is enforced through the static getInstance() method and a private constructor.

Relationships between classes are clearly defined:

* A Game contains a list of Team objects (List<Team>), representing a one-to-many relationship.
* Similarly, a Team contains a list of Player objects (List<Player>), also a one-to-many relationship.
* The Player class, however, does not hold references to its parent Team or Game, reflecting a unidirectional relationship. This design ensures that each player has a unique identifier and can be assigned to a team without directly managing team associations.

The ProgramDriver class serves as the entry point with the main() method and utilizes the SingletonTester to verify the singleton behavior of GameService.

Overall, this design efficiently supports the software requirements by:

* Ensuring uniqueness of games, teams, and players through ID tracking in GameService.
* Enforcing a single game instance using the singleton pattern.
* Structuring the relationships between games, teams, and players in a scalable and maintainable way through object-oriented design principles.

**"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)

**06/08/2025**

I’ve updated each recommendation based on additional research and learning. I also included a system architecture view with a model I created. I researched system architecture components and did my best to apply that knowledge. From what I’ve gathered, most of this game could be developed using C++, as it remains one of the most widely used languages in the game development industry. Additionally, I explored server options and identified some cost-effective solutions, such as renting servers or utilizing peer-to-peer (P2P) network systems. Games like *Minecraft* already use these methods, making them viable options worth considering.

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** | **Server Hosting Capability:** MacOS can host server-based applications, though it is not commonly used in production environments. It supports local development and testing, but large-scale deployment is rare.  **Advantages:** Can run Mac, Windows, and Linux apps side-by-side; user-friendly; consistent UI/UX.  **Disadvantages:** Higher hardware cost and limited hardware configurations.  **Licensing Costs:** Significant, Apple hardware is required, and macOS Server is a paid add-on. | **Server Hosting Capability:** Linux is the most widely used server OS globally.  **Advantages:** Open source, flexible distributions, superior security, and low (or no) cost.  **Disadvantages:** Steeper learning curve; compatibility issues with certain proprietary file formats.  **Licensing Costs:** Typically, none, Linux is free under GNU/GPL licenses. Enterprise-level support (e.g., Red Hat) may incur costs. | **Server Hosting Capability:** Widely used in corporate environments; supports IIS for hosting web applications.  **Advantages:** Native integration with Active Directory and corporate tools; widespread support.  **Disadvantages:** More vulnerable to security threats; less popular in modern web-based game hosting.  **Licensing Costs:** Higher, Windows Server requires licenses that may range from hundreds to thousands of dollars depending on user count and features. | **Server Hosting Capability:** Technically possible using apps on Android, but impractical for scalable deployments.  **Advantages:** Low cost, convenient for local or personal testing.  **Disadvantages:** High vulnerability, limited resources, and reliance on cloud-based solutions for anything beyond basic hosting.  **Licensing Costs:** Generally low; however, additional cloud or app-related fees may apply. |
| **Client Side** | macOS is user-friendly once mastered but is limited to Apple hardware, restricting developer access unless they use Apple devices. This can increase cost and limit team flexibility unless all developers have access to macOS systems. | Linux is cost-effective and offers extensive control and customization. Maintenance is typically lower-effort due to its simplicity. However, developers must handle security independently, as Linux lacks the commercial support provided by macOS and Windows. | Windows is widely available at various price points, offering flexibility for projects with different budgets. It provides extensive technical support and more robust built-in security than Linux. However, developers must be proficient in Windows-specific technologies, and some features may incur additional costs. | Mobile development tools are widely accessible and affordable. However, platform fragmentation (iOS, Android, etc.) means developers often need to tailor apps to specific operating systems, increasing development time and complexity. Mobile environments also lack the full capabilities of desktop systems. |
| **Development Tools** | **Primary Language:** Swift (for native apps), but also supports cross-platform languages like C++, Python, and JavaScript.  **Key Tools:** Xcode and Xcode Cloud, streamline development, CI/CD for Apple platforms.  **Impact on Team:** Requires Apple hardware and macOS expertise.  **Licensing Costs:** Xcode is free; however, Apple Developer Program costs $99/year for deployment. | **Primary Tools:** Docker (for containers and deployment), text editors (Vim, Emacs), and full IDEs (Eclipse, VSCode).  **Impact on Team:** Flexible and scalable; ideal for server-side and open-source web applications.  **Licensing Costs:** Usually none, most tools are free and open-source. | **Primary Language:** C/C++; C# also common for enterprise or game development.  **Key Tools:** Visual Studio, powerful IDE with debugging, version control, and deployment support.  **Impact on Team:** Requires Windows expertise and possibly separate teams for client and server development.  **Licensing Costs:** Visual Studio Community is free; Professional and Enterprise editions incur costs. | **Primary Language:** Java and Kotlin for Android; Swift for iOS. Web-based approach relies on HTML, CSS, JavaScript.  **Key Tools:** VSCode, IntelliJ IDEA, Xamarin for cross-platform development.  **Impact on Team:** Developers must be familiar with mobile platforms and responsive web design.  **Licensing Costs:** Most tools have free versions, though some advanced features or plugins may require payment. |

## 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**: To support the expansion of *Draw It or Lose It* into additional computing environments, Microsoft Windows is the recommended server platform. Windows offers strong cross-platform development support and integrates smoothly with existing Android-based systems, making it ideal for broad deployment.
   1. Seamless integration with Android builds, facilitating multi-platform compatibility.
   2. Support for cross-platform frameworks like Xamarin, Cordova, and React, reducing development redundancies.
   3. Access to a wide range of Microsoft development tools that simplify design, coding, and deployment.
   4. Availability of emulators and testing tools, including Command Prompt, PowerShell, and Ubuntu, to ensure compatibility across environments.
   5. A dominant global market share (~90%) in desktop OS usage, offering a large developer base and strong community support.

Windows provides a scalable, well-supported environment for developing and deploying cross-platform applications, making it a strategic choice for The Gaming Room's future growth.

**Operating Systems Architectures**: The recommended architecture is a hybrid design, as seen in the Windows NT kernel. It combines elements of monolithic, layered, and micro-kernel architecture, offering improved performance, security, and modularity. With fewer layers than a traditional layered model, it’s easier to manage while still allowing flexibility and customization based on client needs. This makes it a well-rounded choice over singular architecture models.

1. **Storage Management**: A centralized, server-based storage solution is recommended to support cross-platform scalability and maintain data integrity. This approach allows for improved performance through features such as automated backups, redundancy, and failover clustering. Centralized storage also facilitates better access control and supports scalability as the user base grows. For a Windows-based platform, Microsoft Azure is the ideal storage management system due to its seamless integration, reliable support, and regular updates.
   1. Azure Cloud: Supports Docker containers and scalable cloud storage.
   2. Flexible Scaling: Easy-to-apply options for growing storage needs.
   3. Cost Efficiency: Affordable without sacrificing performance.
   4. Robust Storage Features: Supports file systems, containers, and blob storage, ideal for managing the client's 200 8MB images.
   5. File Sharing: Offers 1.6GB of storage per user for collaboration.

(<https://azure.microsoft.com/en-us/products/storage/files/#features>)

1. **Memory Management**: The Windows operating system manages memory using both virtual and physical memory allocation. This ensures efficient memory usage for applications like *Draw It or Lose It*. Each process is given its own virtual address space, which provides sufficient memory and improves isolation and performance.

Windows supports disc paging and demand paging, allowing it to adapt to higher memory demands. In addition, tools like Azure Storage, OneDrive, and Visual Studio integrate with the OS to provide cloud-based version control and dynamic memory allocation. These services help optimize performance through garbage collection and resource management, ensuring the application runs efficiently, even under heavy usage.

By following industry-standard best practices and choosing memory-efficient storage containers, developers can further enhance runtime performance and scalability. Windows’ ongoing improvements in memory handling make it well-suited for this project.

1. **Distributed Systems and Networks**: To enable *Draw It or Lose It* to function across various platforms, a cross-platform development framework like .NET MAUI is recommended. This approach streamlines development and ensures consistent functionality across devices without duplicating effort across multiple teams.

For communication between platforms, distributed software and a robust network infrastructure are essential. Using Microsoft Azure Cloud Services enhances connectivity by offering built-in support for App Insights, logging, monitoring, and email alerts. These features help maintain maximum uptime and allow automated responses to system issues, reducing the burden of manual monitoring and enabling the team to focus more on feature development.

To ensure reliability, server infrastructure should be provisioned with scalable capacity, guided by expected user traffic and usage patterns. This supports load balancing and failover mechanisms, helping to minimize downtime and maintain smooth communication across the distributed network, even during outages or high-demand periods.

1. **Security**: Given the sensitivity of user data and the need for secure cross-platform communication, implementing advanced security protocols is essential. While built-in operating system security provides a foundation, it is strongly recommended to supplement it with a dedicated security service like Aura. Aura offers real-time protection for PC, Mac, Android, and iOS devices, along with 24/7 U.S.-based support, ensuring an added layer of defense during data storage, transmission, and access.

To further enhance security, particularly in a Windows and Azure-based environment, the client should utilize Azure App Service with Active Directory through an App Service Plan. This allows:

* 1. Role-based access control (RBAC), limiting user access to data strictly on a need-to-know basis.
  2. IP configuration and restriction, enhancing access control across distributed platforms.
  3. VPN-based storage options, offering encrypted and private communication channels.
  4. Database security features such as SSL enforcement and IP whitelisting, safeguarding data from unauthorized access.
  5. Data encryption at rest and in transit, protecting user information from breaches across all connected systems.

Finally, company-wide awareness of password hygiene, multi-factor authentication (MFA), and standardized authorization protocols is crucial. These practices reduce internal vulnerabilities and strengthen the overall security posture of the application across platforms.

(<https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-database-security>)