

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 | 01/31/2025 | Andrew Corrigan | Project One |

**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 is in need of a web-based version of its popular game, “Draw It or Lose It,” so it can reach multiple platforms and accommodate larger numbers of players. The core challenge is ensuring the game can handle multiple teams and players while enforcing unique names and maintaining only one instance of the game in memory at any given time. By creating a single GameService class that manages all games, teams, and players, we can ensure that the application only instantiates the game environment one time. In addition, each domain object (game, team, or player) inherits from a shared Entity base class to avoid duplicated fields. This design creates a streamlined and scalable structure, so developers can easily extend the software to new platforms without sacrificing performance or consistency.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

- The application must maintain only one active instance of the game in memory.

within their respective contexts. The system checks for duplicates by iterating over existing objects whenever a new one is added.

- Multiple users and platforms (web and mobile) may access the application simultaneously, so the design must be thread-safe and efficient.

- As the game user base grows, the architecture must easily support additional features and high traffic.

## [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 diagram shows how the game application’s classes are structured and interact. At the center is an Entity class that acts as a base for the Game, Team, and Player classes by providing shared attributes and behaviors (namely id and name). From there, a Game can have multiple Teams, and each Team can have multiple Players. This design reflects inheritance, since Game, Team, and Player all extend Entity rather than duplicating those common fields. It also demonstrates encapsulation, with private fields and getter methods ensuring that data access is controlled.

**"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** | MacOS has a Unix foundation, allowing compatibility with many Linux-based tools and frameworks. It provides a stable environment with built-in development utilities like Homebrew and Xcode Command Line Tools. It is excellent for local testing and smaller-scale servers and offers a consistent environment for teams already using macOS. However, Apple hardware is expensive, making it less common for large-scale deployments. Few hosting providers support macOS servers, and scalability is limited compared to Linux or Windows. | Linux is the most commonly used system for web hosting, dominating cloud and enterprise environments due to its scalability, security, and flexibility. It is free or low-cost, supports Docker and Kubernetes for easy scaling, and is preferred by most cloud providers. However, Linux requires strong technical knowledge to manage, and the fragmentation across distributions can introduce compatibility challenges if not carefully handled. | Windows Server integrates with Microsoft technologies like .NET, IIS (Internet Information Services), and Active Directory, making it useful for organizations already using Microsoft products. It provides strong vendor support and enterprise tools like SQL Server, but licensing costs are high. Windows requires more system resources than Linux, making it less efficient for large-scale hosting. | Mobile devices (iOS, Android) are not built for hosting, but they serve as essential client endpoints. Some tools, such as Node.js on Android via Termux, allow small-scale hosting, but this is not practical for large-scale use due to limited resources, battery dependency, and network restrictions. |
| **Client Side** | Mac users rely on Safari, which requires specific optimization compared to Chromium-based browsers like Chrome or Firefox. While macOS provides a stable and secure experience, Safari’s differences in rendering may require additional testing. Mac hardware is expensive, but once purchased, there are no additional license costs for accessing web applications. | Linux users primarily use Chrome and Firefox, which ensures strong compatibility with web-based applications. However, Linux desktop users represent a smaller portion of the market, so developers may need to conduct extra testing to ensure proper performance across distributions. The open-source nature of Linux gives users more control, but variations across different versions can create minor inconsistencies. | Windows is the most widely used desktop operating system, meaning the game must be optimized for Chrome, Edge, and Firefox. Windows offers strong web-based support, but it is more vulnerable to malware and security threats compared to macOS and Linux, requiring extra precautions when handling user data. | Mobile users are a major audience, making iOS and Android support crucial. A responsive web application ensures accessibility, but a native mobile app could offer better performance, smoother animations, and offline features. Mobile apps require separate development for iOS (Swift, Xcode) and Android (Kotlin, Java, Android Studio), but cross-platform frameworks like React Native or Flutter can help streamline development. |
| **Development Tools** | Mac supports development tools like Swift, Objective-C, JavaScript, and Node.js. Xcode is the main IDE for iOS/macOS development and is required to publish iOS apps. Xcode is free, but developers must use Mac hardware, and publishing to the App Store requires an annual $99 fee. Mac also supports cross-platform tools like Visual Studio Code, JetBrains WebStorm, and Docker, making it useful beyond iOS development. | Linux is a flexible development environment that supports JavaScript, Python, Node.js, Java, and C++. It is commonly used for backend development and web applications, with tools like Visual Studio Code, Eclipse, and IntelliJ available. Docker and Kubernetes make it easy to build and deploy applications efficiently. Since Linux is open-source, most development tools are free, making it the most cost-effective development platform. | Windows is commonly used for .NET and C# development, with Visual Studio as the primary IDE. It also supports Node.js, Python, and Java for web applications. Windows provides a strong development environment, but enterprise versions of Visual Studio require licensing fees. Windows is also widely used for game development, supporting Unity, Unreal Engine, and DirectX. Docker for Windows is available but less seamless compared to Linux. | Mobile development requires separate tools for iOS and Android. Android Studio (Java, Kotlin) is needed for Android development, while Xcode (Swift, Objective-C) is required for iOS. React Native and Flutter allow cross-platform development, reducing time and cost. Publishing to Google Play requires a one-time $25 fee, while publishing to the App Store requires an annual $99 fee. |

## 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**: After comparing several options, Linux stands out as the most reliable choice for hosting “Draw It or Lose It.” It’s well-known for being stable, secure, and cost-effective, all of which are crucial for an online game. Most major cloud providers fully support Linux, and there’s a huge community that creates and maintains free tools, making it easier to find the help you need. Going with Linux is going to give us a simpler setup with more efficient use of system resources, and much lower long term costs compared to some of the competition.
2. **Operating Systems Architectures**: Linux relies on a monolithic kernel architecture, so its core functions like device drivers and file system handlers get bundled together in one central part of the operating system. Practically this design makes Linux both powerful and responsive, even under heavy traffic. At the same time, Linux supports modular extensions, allowing you to add or remove certain features (like specialized device drivers) without having to rebuild the entire kernel. This balance between flexibility and performance is one reason Linux is so popular in large-scale web hosting.
3. **Storage Management**: For data storage covering everything from user profiles and team details to overall game history, relational databases like MySQL or PostgreSQL are porbably going to be the best fit. They're going to excel at keeping data accurate and consistent, which is especially important in a multiplayer setting. If more flexibility is needed for unstructured data, you can pair your main database with a NoSQL option like MongoDB. Because Linux works seamlessly with these open-source databases, setting them up and keeping them running smoothly is usually straightforward.
4. **Memory Management**: Linux offers robust virtual memory management, using techniques such as demand paging and swap spaces to maintain steady performance as your player base grows. Essentially, the system moves data you’re not actively using out of main memory so you don’t run out of resources when traffic spikes. To ensure there’s only one active instance of the game at any time you’ll want application-level controls, a “game manager” that keeps an eye on memory usage and prevents multiple versions of the game from starting. This setup helps keep everything running smoothly and efficiently.

**Distributed Systems and Networks**: When you want players on different devices to interact in real time, a distributed services model is ideal. Under this approach, the core features of the game are made available through an API-driven setup, typically using REST or WebSocket endpoints. This means any client platform can talk directly to the server. In order to handle high traffic, load balancing tools (like NGINX) spread incoming requests across multiple servers, avoiding bottlenecks or crashes. If you run multiple instances of the application, this also provides redundancy so if one instance fails, others keep the game accessible. Many cloud providers already offer built-in scaling and failover features, making the system more resilient overall.

1. **Security**: Protecting user data is a top priority, especially when it comes to personal information. One of the first steps is encrypting all data in transit, typically with HTTPS/TLS, so it stays confidential between the user’s device and the server. At rest, sensitive information like passwords should be encrypted or hashed, and full-disk encryption can provide an extra layer of safety. Strong authentication methods (like OAuth 2.0) help ensure that only authorized users can log in, while things like regular system updates, removing unnecessary software, and configuring firewalls can all reduce potential security gaps. Finally, real-time monitoring and alerts allow for suspicious activity like repeated failed logins to be identified and responded to quickly, giving you an additional safeguard for your game and its players.