

# 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 | 03/23/25 | Johnny Chen | Summary/Design Constraints/Domain Model changes. |

**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 has requested the development of a web-based version of their game "Draw It or Lose It," which is based on the 1980s television game show "Win, Lose or Draw." We propose developing a client-server web application that implements a Singleton pattern to ensure only one game instance exists in memory, a robust domain model with unique identifiers for games, teams, and players, RESTful API with webhook integration for real-time client-server communication, and responsive design for cross-device compatibility, all built with modern web technologies that prioritize scalability, performance, and user experience to establish a foundation for both current requirements and future enhancements.

## Requirements

The client, The Gaming Room, has specified the following business and technical requirements for the "Draw It or Lose It" game application:

1. The game must support one or more teams participating in gameplay.
2. Each team must be able to have multiple players assigned to it.
3. Game and team names must be unique within the system, allowing users to verify name availability during team creation.
4. Only one instance of the game can exist in memory at any given time, achieved through unique identifiers for each game, team, and player instance.
5. The game must consist of four one-minute rounds of play.
6. Drawings must render progressively, reaching full completion at the 30-second mark of each round.
7. If the active team fails to guess correctly before time expires, other teams must have one 15-second opportunity to guess the puzzle.
8. The system must render images from a pre-existing library of stock drawings rather than having players create drawings in real-time.

## [Design Constraints](#_2et92p0)

Developing Draw It or Lose It as a web-based distributed application comes with several key design constraints. Since the web uses a stateless HTTP protocol, real-time game updates require tools like WebSocket’s and strong state management. The game must work smoothly across different browsers and devices, so responsive design is essential. Network delays and connection issues can affect gameplay, so the system needs to handle timing carefully on both the client and server sides. Security is also important, with strong authentication and data validation needed to protect users and prevent cheating. To support many users at once, the system must be scalable with efficient resource use and load balancing. Finally, only one active game instance should exist in memory, so a centralized state system or server-side singleton pattern must manage all player interactions reliably.

## [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 shows how the main parts Game, Team, and Player are connected in a clear and organized way. All three classes inherit from a common Entity class, which holds shared properties like ID and name. This use of inheritance helps reduce repeated code and keeps things simple. The Game class contains a list of teams, and each Team contains a list of players, showing a one-to-many relationship that reflects the game structure described in the requirements. The GameService class uses the Singleton pattern, meaning only one instance of the service exists in memory at a time, which meets the requirement that only one game can be active at once. It also manages unique IDs for games, teams, and players to ensure no duplicates. The diagram uses encapsulation by keeping attributes private and providing public methods to interact with them. Overall, the design uses core object-oriented principles like inheritance, encapsulation, composition, and modularity to create a system that is easy to manage, scalable, and meets the client's needs 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** | MacOS can host web applications using tools like Apache, Nginx, and Docker, but it is not commonly used as a production server OS. The limited availability of Apple server hardware and its higher cost makes it a less scalable solution. Licensing costs are typically bundled with hardware. For small scale internal deployments or prototyping, it can be useful, but for enterprise level hosting, it's not ideal. | Linux is widely used for web application hosting due to its open source nature, strong community support, high performance, and robust security features. It supports popular web servers (Apache, Nginx) and containerized environments (Docker, Kubernetes). Licensing costs are minimal or nonexistent, making it ideal for scalable and cost-effective deployment. | Windows Server can reliably host web applications and integrates tightly with the .NET ecosystem and IIS for deployment. It is suitable for enterprise environments, especially where Windows-based infrastructure is already in use. However, it does have licensing fees, and resource usage is typically higher than Linux. | Mobile devices are not designed to function as web servers. While apps like Termux or custom setups can simulate a local server environment, they lack the power, stability, and scalability for web hosting. They are best suited as client endpoints communicating with the hosted web application. |
| **Client Side** | MacOS users access the application through modern web browsers like Safari, Chrome, and Firefox. Development requires access to mac hardware for proper UI/UX testing. Apple’s strict hardware requirements can increase development and testing costs but, macOS is compliant with standards, and HTML5 based applications will function well with responsive design. | Linux users primarily use Firefox and Chrome, which support modern web standards. Although Linux has a smaller desktop market share, web apps that follow W3C standards should work seamlessly. Development costs are minimal, and tools are widely available. Testing on Linux ensures broader compatibility but may be lower priority due to fewer end users. | Windows is the most popular desktop OS and supports all major browsers, making it a high priority platform for testing. Developers need to ensure compatibility with multiple browser engines. Development is cost effective and widely supported. Teams already familiar with Windows will find it easier to debug and deploy cross platform browser applications. | Mobile users access the application via mobile browsers (Safari on iOS, Chrome on Android). Responsive design is critical, and additional testing is needed to account for different screen sizes, touch interactions, and performance. Time and cost increase with the need to optimize for slower mobile networks and device variability. |
| **Development Tools** | MacOS development supports tools like Xcode (necessary for iOS builds), VS Code, and JetBrains IDEs. Mac is required for iOS deployment, meaning a portion of the team must use mac hardware. This increases hardware cost. However, tool licenses are typically free or low cost. Full-stack development is feasible, especially for hybrid apps. | Linux supports a wide array of open source tools, including Eclipse, NetBeans, IntelliJ, and CLI utilities. It’s a favorite among backend developers for its scripting and automation capabilities. Developers can easily manage environments with package managers. No OS license costs and broad tool compatibility make Linux development highly efficient. | Windows supports robust development tools like Visual Studio, VS Code, and JetBrains IDEs. It’s ideal for enterprise grade application development, especially when using C# or .NET technologies. Licensing for Windows and premium IDEs can add cost, but the ecosystem is mature and well documented. | Mobile app development typically requires Android Studio for Android and Xcode for iOS. Cross platform frameworks like Flutter or React Native can unify development, reducing time and cost. Emulators and device simulators assist with testing. Mobile development requires expertise in touch interface design, platform guidelines, and integration with native APIs. |

## 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 growth of Draw It or Lose It across different platforms, Linux-based servers are a strong choice. Linux is known for its stability, performance, and security, and it’s widely used in web application hosting. Its open source nature also makes it cost effective, which is ideal for scaling the game as the user base grows.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel architecture, meaning that key system functions operate together in one space, resulting in efficient system performance. It also works well with tools like Docker, which can isolate game sessions and improve resource management. This is especially useful in a distributed game environment where multiple players are active simultaneously.
3. **Storage Management**: Given the games use of large image files and player data, a hybrid storage solution is recommended. Frequently used assets can be stored locally for quick access, while additional assets can be pulled from cloud storage services such as AWS S3 or Google Cloud Storage. For handling player data, both relational databases (like PostgreSQL or SQLite) and NoSQL solutions (like MongoDB or DynamoDB) are recommended. NoSQL databases are especially effective for distributed systems due to their ability to horizontally partition data across servers, allowing the application to scale efficiently.
4. **Memory Management**: Performance is critical in a game like Draw It or Lose It, especially when loading large images. Linux provides strong memory management features such as paging and virtual memory. On the application level, techniques like lazy loading, object pooling, and caching with something like Redis can significantly improve responsiveness. Redis, as an in-memory data store, can help quickly retrieve frequently used data, reducing latency during gameplay.
5. **Distributed Systems and Networks**: The game will need to operate across multiple platforms, requiring a distributed system architecture. Standard communication can be handled through REST APIs, while WebSockets or gRPC can support real time interactions. To ensure decoupled and scalable communication between services, a message broker like RabbitMQ or Apache Kafka can be used. This allows different components to send and receive messages asynchronously, which improves system resilience under heavy traffic. In addition, tools like load balancers and Kubernetes can manage scalability and help the system recover from outages. Secure communication protocols like HTTPS and TLS should be used throughout.
6. **Security**: To ensure user data is protected across all platforms, the system should implement several key security practices. Authentication can be handled using JWT (JSON Web Tokens), which allows for secure, stateless sessions. Authorization should be enforced through Role Based Access Control (RBAC) to ensure users only have access to the features they’re permitted to use. All data should be encrypted both in transit using TLS and at rest to prevent unauthorized access. Additionally, strong input validation must be in place to protect against injection attacks and other forms of malicious input.