

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 3/20/2025 | Alayna De Valk | Executive Summary, Design Constraints, Domain Model |
| 2.0 | 4/1/2025 | Alayna De Valk | Client Side/Server Side/Development Tools Evaluation |
| 3.0 | 4/14/25 | Alayna De Valk | Recommendations |

**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 seeks to develop a web-based version of their existing Android game, Draw It or Lose It, making it accessible across multiple platforms, including desktops, laptops, and mobile devices.

The new application must support multiple teams, with each team consisting of multiple players.

Game and Team Names must be unique. Using an iterator pattern, the program can check whether a name is already in use before assigning it to a team.

A singleton pattern will allow us to ensure that only one instance of the game will exist in memory at any given time. This way, we can create unique identifiers for each instance of game, team or player.

## 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)

Cross-Platform Compatibility

The game must be accessible on desktops, laptops, and mobile devices across different operating systems and web browsers. To achieve this, the front-end will be developed using responsive web technologies such as HTML5, CSS, and JavaScript frameworks, ensuring a seamless user experience regardless of screen size or input method.

Network Reliability

Since the game operates in a distributed web environment, network latency, packet loss, and disconnections could impact real-time gameplay. Implementing WebSockets or a similar real-time communication protocol will ensure smooth data transmission, while error-handling mechanisms will provide graceful reconnection in case of temporary network failures.

Scalability and Concurrent Users

The system must support multiple concurrent players and teams without performance degradation. To manage high user loads efficiently, the application will utilize load balancing, cloud-based hosting, and database indexing, ensuring the game can scale dynamically and maintain a responsive experience for all players.

## [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 for The Gaming Room outlines the structure and relationships between classes needed to support the web-based game Draw It or Lose It. At the center of the design is the Entity class, which acts as a base class for Game, Team, and Player. This base class contains shared attributes such as id and name, which promotes reusability and consistency across the application. Each subclass inherits these common properties, reducing redundancy and improving code maintainability.

The Game class manages a list of Team objects, and each Team manages a list of Player objects, illustrating a clear hierarchy and logical relationship between game components. The GameService class serves as a controller, responsible for creating and retrieving game instances. It follows the Singleton design pattern to ensure that only one instance of the game service exists in memory at any given time, which aligns with the software requirement that only one instance of the game should exist. The use of static ID generators within GameService also ensures each entity—game, team, and player—receives a unique identifier.

This UML diagram demonstrates all four fundamental object-oriented programming (OOP) principles. Encapsulation is shown by private attributes and public getter methods that protect internal data. Inheritance is applied through the shared Entity base class, promoting code reuse across related classes. Polymorphism is supported via methods like toString() that can be overridden in subclasses to provide custom behavior. Abstraction simplifies the system by exposing only the essential features of each class, such as adding teams or players, while hiding the underlying complexity.

All of this works together to build a flexible and easy-to-manage system that meets the client’s needs. It keeps things organized, makes sure names stay unique, and supports having multiple teams and players while sticking to one game instance in memory.

**"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** | While Macs are great for development, they aren’t usually used as web servers in production. They’re Unix-based like Linux, which is helpful, but most hosting providers don’t offer Mac servers, and macOS licenses and hardware are more expensive. It can technically be done, but it’s not ideal for a scalable server setup. | Linux is one of the best options for hosting web applications. It’s free, stable, and widely supported. It works well with common server tools like Apache and NGINX and is used by most cloud hosting providers. It’s also highly customizable, which makes it a solid choice for scaling up to thousands of users. | Windows servers are available and work well with Microsoft tools. They use IIS (Internet Information Services) for hosting websites. While they can be easy to manage if you're familiar with the Windows environment, they come with higher licensing costs and tend to use more system resources compared to Linux. | Mobile platforms like iOS and Android aren’t designed to act as servers. They’re built to access services, not host them. You wouldn’t use a phone or tablet to run a scalable web application, so they aren’t suitable for this part of the project. |
| **Client Side** | Mac users can access the app through browsers like Safari or Chrome. As long as the application is responsive and built using standard web technologies like HTML, CSS, and JavaScript, it should work smoothly. Just a little browser testing is needed to make sure everything displays correctly. | Linux users will mostly use browsers like Firefox or Chrome to access the app. Since these are modern browsers, the experience should be the same as on other platforms. Again, responsive design and proper testing will ensure it works across different screen sizes and systems. | Windows users typically browse with Chrome, Firefox, or Edge. Like with Mac and Linux, building the app as a web-based interface means you only need to focus on supporting modern browsers. This keeps things simple and avoids building a separate desktop version. | The application will need to be responsive and mobile-friendly so it displays well on smaller screens. Users can open it in Safari (iOS) or Chrome (Android). You’ll want to make sure buttons and inputs are touch-friendly, and that the layout adjusts properly across different devices. No need for a full native app unless the client requests it in the future. |
| **Development Tools** | Mac is great for both frontend and backend development. Tools like IntelliJ IDEA, Visual Studio Code, and Eclipse work well here. You can easily run Java for the server-side and use standard web tools for the frontend. Most tools are free or offer a free version. | Linux is widely used by developers for backend work, especially when working with Java and web servers. It's free to use and comes with strong support for tools like Eclipse, Maven, Git, and other open-source resources. Developers can do just about everything they need on Linux without extra licensing costs. | Windows works well for development too and supports all the same major tools—like IntelliJ, Eclipse, and Visual Studio Code. Some Unix-based tools might need extra setup, but most development environments work smoothly. Licensing depends on which version of the tools you use, but there are free options available. | For web-based mobile access, the same tools for frontend development apply: HTML, CSS, JavaScript, and frameworks like React. For native development, you’d use Android Studio or Xcode, depending on the platform. While mobile devices can be used to test web applications, they aren’t practical environments for full-scale software development due to limited processing power, screen size, and lack of support for most development tools. But since the client wants a browser-based solution, the development team can stick with one frontend and test it across devices instead of building two separate mobile apps. |

## 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**: For hosting the web-based version of Draw It or Lose It, I recommend using a Linux-based platform. According to Khanna (2025), Linux is often preferred for its reliability, cost-effectiveness, and compatibility with cloud environments. Linux is reliable, efficient, and doesn’t come with expensive licensing fees like Windows Server does. It works well with many common open-source tools and most major cloud providers offer great support for Linux environments. With the client’s need for scalability and performance, Linux just makes the most sense.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel, meaning all the core operating system functions are built into a single kernel. This setup can help with performance, especially when the game needs to respond quickly to things like live player actions or real time image rendering. What’s also helpful is how modular Linux is, meaning you can load or unload system components based on what the application actually needs. This can help keep things lightweight and running smoothly as more users join, enabling streamlined performance in dynamic environments (ARMO, n.d.). In terms of application architecture, a stateless client-server model is recommended. This approach allows the frontend client to run in the browser while the backend handles game logic, data management, and user authentication via RESTful APIs.
3. **Storage Management:** Since the game includes a large image library and will likely collect user and gameplay data over time, I recommend using a cloud storage service like Amazon S3 or Google Cloud Storage. These platforms offer scalable, object-based storage that allows files to be organized by how frequently they're accessed, so high-demand content remains in fast-access storage while older assets can be archived at a lower cost (Amazon Web Services, n.d.; Google Cloud, n.d.). Both services also support built-in options for backup, automatic cleanup, and lifecycle policies, which help keep storage organized and prevent overuse. They are widely adopted in web development because of their strong performance, security, and seamless integration with modern backend infrastructure (Amazon Web Services, n.d.; Google Cloud, n.d.).
4. **Memory Management**: To keep the game running fast and smooth during rounds, the game should only load what’s needed for the current round and let go of that data once it’s no longer needed. For Draw It or Lose It, Linux’s memory management helps keep gameplay responsive by using demand paging and virtual memory to load only the essential game assets, like the current round’s image, into RAM. Tools like htop and vmstat can be utilized during development and live hosting to monitor memory usage, helping to identify and prevent memory leaks or performance issues (GeeksforGeeks, 2023a; GeeksforGeeks, 2023b).
5. **Distributed Systems and Networks**: Since the game is expected to run on multiple platforms and support a large number of users, a distributed system setup is the way to go. The backend should be stateless, using RESTful APIs or WebSockets to communicate in real time. A load balancer can help spread traffic across different servers, preventing one server from getting overwhelmed. By storing data in the cloud and syncing it in real time, players can enjoy a smooth experience across any device. The addition of automatic failover and regular health checks will help the game stay online, even if something crashes in the background.
6. **Security**: Security needs to be built in from the start of the application. Linux provides strong tools for managing user access, file permissions, and firewall protection. On the application side, all traffic should be encrypted with HTTPS, and passwords should be securely hashed. Using something like OAuth 2.0 or a trusted identity provider could help manage logins safely. Role based access should be enforced so that only authorized users can perform certain actions. Common security risks like XSS or SQL injection can be avoided with good input validation and careful coding. Logging and monitoring tools can also help catch anything suspicious early.

**References**  
  
Amazon Web Services. (n.d.). *What is Amazon S3?* AWS Documentation. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>

ARMO. (n.d.). Linux Kernel: Core Functions, Architecture, and Customization. <https://www.armosec.io/glossary/linux-kernel/>

GeeksforGeeks. (2023, June 20). *Using htop to monitor system processes on Linux*. https://www.geeksforgeeks.org/using-htop-to-monitor-system-processes-on-linux/

Google Cloud. (n.d.). *Cloud Storage overview*. Google Cloud Documentation. <https://cloud.google.com/storage>

Khanna, A. (2025, February). Understanding the Kernel Process in Linux vs Windows. LinkedIn. <https://www.linkedin.com/pulse/understanding-kernel-process-linux-vs-windows-abhinay-khanna-sjl0c>