

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 | 05/25/2025 | Patricia Randol | Focus on expanding the Executive Summary, Design Constraints, and Domain Model analysis. |

**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 expanding its Android-based game, Draw It or Lose It, into a web-based multi-platform environment. The new system must accommodate multiple teams, ensure unique game and team names, and enforce a single active game instance at any given time. Additionally, each game must allow multiple players per team while maintaining real-time responsiveness.

## Requirements

The Gaming Room has outlined specific business and technical requirements for the web-based expansion of Draw It or Lose It. These requirements will shape the software design and impact development decisions.

Business Requirements

Multi-Team Gameplay – The game must allow multiple teams to participate simultaneously.

Multiple Players per Team – Each team must have multiple players assigned.

Unique Game and Team Names – Users must be able to check whether a name is in use when choosing a team name.

Cross-Platform Accessibility – The game must be playable on multiple platforms beyond Android, including web browsers.

Technical Requirements

Singleton Pattern Enforcement – Only one instance of the game can exist in memory at a given time.

Unique Identifiers – The system must generate distinct identifiers for each game session, team, and player.

Real-Time Rendering – Drawings should progressively render in each round, ensuring visual clues develop over time.

Turn-Based Guessing Mechanism – After the primary team fails to guess, opposing teams must be given a 15-second guessing window.

Database-Driven Validation – A backend database must efficiently manage game states, name uniqueness, and player assignments.

Scalable Architecture – The game should be designed to handle high user traffic, ensuring performance efficiency in distributed environments.

## [Design Constraints](#_2et92p0)

Design Constraints

The web based, distributed nature of Draw It or Lose It has imposed a number of design constraints on the developing process.

1. Concurrency & Synchronization

The game should provide support for more than one player and more than one team to join the game play at the same time, and a well working session management is key here.

We need to take care of concurrency control mechanisms including thread synchronization and state management with Redis to avoid race conditions among game state updates.

2. Singleton Pattern Enforcement

There should be only one game held in memory at a time.

Implementing a Singleton pattern to prevent more than one active game, but with tight access control to avoid instance creation by mistake.

3. Unique Name Validation

Game and team names information must be unique, thus efficient database indexing and validation is crucial.

Performing lookups can be an expensive operation unless it is optimized.

4. Scalability & Performance

The number of interacting users which the system can accommodate must increase, and the response time should not degrade.

Traffic can be distributed efficiently using load balancing techniques and microservice architecture.

By implementing horizontal scaling, growing player activity can be smoothed out.

5. Real-Time Display & Application

The game depends on timed drawing reveals and player guesstimates, demanding low-latency interactions.

Some form of WebSocket’s or server-sent events (SSE) need to be set up to update the game in real time without any noticeable delay.

6. Cross-Platform Compatibility

The game shall function seamlessly across devices, requiring support for Chrome (Windows/Android), Safari (macOS/iOS), Firefox (Linux), and Edge (Windows).

As of 2025, Chrome leads with 62% global browser usage, followed by Safari at 18% (StatCounter, 2025). Ensuring compatibility with these engines-Blink and WebKit- is vital for accessible gameplay.

7. Security & Data Integrity

Private user data (e.g., group association, gaming records) needs to be protected from unauthorized access.

The use of OAuth for authentication and data encryption complies with the best security practices.

Implications for Development

Architectural Design: The concurrency and scalability issues can be dealt with using modular microservices.

A Real-Time System Optimized: WebSocket’s provide an even more responsive gaming experience.

Platform Agnostic: Front-end with React and back-end with Node. js for back end allows cross platform support.

## [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)

Analysis of Domain Model and UML Class Diagram, the UML class diagram of Draw It or Lose It gives a structured overview of the game’s main entities and their relations. Here's an overview of how the class is structured and an indication of how OOP concepts are used to satisfy The Gaming Room's software needs in the most effective way.

Relations of Production and Class Structure Below is the UML diagram of the main classes:

Game Class

Serves as the "brain" for our game logic.

This is due to the Singleton design pattern, and it will help us to have only one running game object at a time.

Oversee the game across rounds as well as drawing rendering and interaction with teams.

Team Class

Represent an array of players in the game.

Team names are unique to ensure that no two teams register with the same name.

It has an association with Player class (each team has many players).

Player Class

Defines specific players on teams.

Stores player information, like the name and the role to play.

It is composed of the Team class to ensure that Players are on a Team.

Entity (Base Class)

It is a parent class of Game, Team and Player where all the fields like UUID, timestamp, and common properties are available.

Supports inheritance so that multiple entities can be reused.

Object-Oriented Concepts of the UML The UML diagram already includes some important OOP features that allow the system to be effective, as well as easily expandable:

Encapsulation

Every class is so self-contained, dealing only with its properties and behaviors, and not allowing other classes to directly access their data.

Example: The Game class manages the game and hides the inner workings as implementation details.

Inheritance

The Game, Team, and Player classes all derive from Entity, which reduces code duplication and hierarchical structure.

This simplifies maintenance and ensures better scalability.

Polymorphism

The framework supports the ability to build such implementations dynamically, such as being able to add and remove players from a team or adding other types of game interactions.

Example: The gameplay rules may be adjusted depending on the teams and on the players’ actions.

Singleton Pattern (GameClass)

Only one instance of a game is ever online simultaneously, removing incompatibilities and saving resources.

Example- The Game class manages game sessions through a private constructor and static access methods.

Composition (Team & Player)

A Team has several Player s which represents a strong coupling.

Teams have their players cascade when the team itself is deleted.

How Do These Principles Meet the Requirements of Software?

Unique Game & Team Names Validation & Database indexing on the Team class.

Team & Player Management → Done through the composition relationships of Team and Player.

Singleton in the Game is implemented by the Singleton pattern in the Game class.

Support Scalable System Design → Decomposed with inheritance and encapsulation and it supports polymorphism, so extendable.

"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 must work together.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Web applications can be developed for the Mac OS and hosted on a Mac. However, a Mac server is not as commonly used as the Windows server; there is less enterprise support. It provides the stability of Unix, without all the rich hosting tools that come with Linux. | Linux Most used for web servers because of performance, security, and scalability. It is compatible with Apache, Nginx, cloud hosting services, such as AWS and Rackspace, which is best for large-scale deployment. | Windows Server is a popular choice among businesses and facilitates IIS (Internet Information Services). The second is more resource-dependent and has licensing costs associated with it | Mobiles are not the right choice for hosting web applications, but they can connect clients with remote servers. Direct Hosting is not feasible due to performance limitations |
| **Client Side** | Developing for macOS calls for a knowledge of Apple’s frameworks such as Swift and Xcode. "Support for Mac users entails Safari and macOS security compliance. | For Linux users, support for open source, which means web apps need to support Firefox, Chrome, and system variants such as Ubuntu. You need to know Linux package management. | Windows desktop apps want to work with Edge, Chrome, Windows security protocols. It's easier to develop as industry has adopted it, but you must test it on multiple versions. | Mobile browsers (Chrome for Android and Safari for iOS) handle rendering differently due to the underlying engines (Blink vs. WebKit). Developers must test mobile UI responsiveness and input behaviors to ensure seamless player interaction. Browser popularity data informs where QA efforts should focus, with Chrome dominating mobile use globally. |
| **Development Tools** | Swift, Objective-C, and development with Xcode are all must for Mac apps. Node is what most people use to build Web applications. js and that we would use JavaScript, and Docker to deploy. | Python, PHP, JavaScript, and Java all form the backbone of Linux development. Developers work with VS Code, Vim, or various JetBrains tools, and servers powered by Apache, Nginx, or cloud services. | Development with Windows-based platform consists of: NET, C#, and Visual Studio. JavaScript, Node. js, and IIS for deployment. | Evaluation → Development Tools (Clarify development location)  Development occurs on desktop platforms (Windows/macOS/Linux), using tools such as VS Code, Android Studio, or Xcode. While apps are built for mobile phones, the development itself doesn’t take place on mobile devices due to hardware limitations. These can be anything from development environments (e.g., Xcode, Android Studio) to cross-platform frameworks such as Flutter or React Native. |

## Recommendations

Analyze the characteristics and techniques specific to various systems architectures and make recommendations to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: To support scalability, performance, and cross platform communication, a **Linux-based cloud environment** (e.g., Ubuntu on AWS EC2 or Google Cloud Compute Engine) is recommended. Linux provides robust server-side tools, efficient resource management, strong community support, and compatibility with containerized applications- making it ideal for hosting Draw It or Lose It.
2. **Operating Systems Architectures**: The platform should adopt a microservice based architecture using Linux containers (Docker) orchestrated through Kubernetes.

This architecture allows each game subsystem (authentication, gameplay, matchmaking,

scoring) to operate independently, enabling better maintainability, scaling, and fault

isolation. A decoupled system also makes integration with CI/CD pipelines more

seamless for faster deployments.

1. **Storage Management**: The recommended storage stack includes:

PostgreSQL for structured, relational data such as team names, game sessions, and

player profiles.

Redis (in-memory data store) for real-time game state caching to ensure low latency.

Cloud object storage (e.g., AWS S3 or Google Cloud Storage) for storing media assets

like game drawings and avatars.

These systems provide ACID compliance where needed and high-speed memory access for live gameplay responsiveness.

1. Memory Management: The Linux platform uses paging, segmentation, and memory-efficient scheduling techniques. When paired with Node.js and containerization:

JavaScript’s garbage collector handles heap memory during game interactions.

Containers allow resource limits and isolation (e.g., memory caps), ensuring that

individual services (like drawing rendering) don’t exhaust system RAM.

Redis reduces pressure on the backend by offloading temporary but performance-critical state information.

1. **Distributed Systems and Networks**: Draw It or Lose It will rely on WebSocket for real-time, bidirectional communication between devices and servers. These channels allow

events (drawing, guesses) to sync instantly across clients. To ensure resilience:

Load balancers and auto-scaling groups will distribute incoming traffic across nodes.

Content Delivery Networks (CDNs) can cache and deliver static assets globally to reduce load times.

The system will include graceful degradation strategies, such as fallback polling, in case of temporary outages.

1. **Security**: Security is embedded across the entire architecture:

**OAuth 2.0** with encrypted tokens (e.g., JWT) will manage authentication.

**Role-Based Access Control (RBAC)** ensures users only access appropriate game features.

All traffic will use **SSL/TLS** encryption.

Regular **security audits** and **dependency vulnerability scans** will be conducted.

Game logs and user activity will be monitored and secured under privacy-compliant policies (e.g., SOC2, GDPR if applicable).

**References**

StatCounter. (2025). *Browser market* *share worldwide*. Retrieved June 20, 2025, from <https://gs.statcounter.com/browser-market-share>

OWASP Foundation. (2023). *Authentication cheat sheet*. <https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html>

Stack Overflow. (2024). *Developer survey results 2024*. <https://survey.stackoverflow.co/2024>

JetBrains. (2025). *The state of developer ecosystem 2025*. <https://www.jetbrains.com/lp/devecosystem-2025>