

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 | 05/23/25 | Sabrina Shaver | Provided an executive summary of the project, identified design constraints, addressed the domain model, evaluated available operating systems for development, and made my recommendations. |
| 2.0 | 06/08/25 | Sabrina Shaver | Revised and expanded on the evaluation and recommendations. |
| 3.0 | 06/22/25 | Sabrina Shaver | Revised and expanded on recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is expanding its Android game, *Draw It or Lose It*, into a web-based, cross-platform application. This game challenges teams to guess phrases based on progressively rendered stock images over the course of four timed rounds. To support the successful transition to a web-based environment, the software design needs to implement some core requirements, including the following: multiple teams with unique names, multiple players per team, enforcement of name uniqueness, and a restriction of having only one active instance of the game in memory at a time.

To address this problem, we propose a solution that utilizes proven software design patterns. The Singleton pattern will ensure that only one instance of the game exists at a time, preventing conflicts in memory. To streamline navigation through player and team lists, we will also implement the Iterator pattern, which allows for traversal of collections without exposing their internal structure. Each game, team, and player will be assigned a unique identifier to ensure distinct references and support organized handling of data.

This approach will support a modular and scalable software foundation, making it easier to further expand the game as needed in the future. Before starting development, it is important that The Gaming Room prioritizes the setup of the development environment and making of platform-related decisions. The proposed design balances immediate functionality with long-term adaptability, and we recommend proceeding with the prototype and testing phase to begin bringing *Draw It or Lose It* to a broader audience.

## 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 following constraints will impact how the software is structured, how it performs, and how it scales in a multi-user environment. Below are the primary design constraints, as well as their implications for development:

1. **Only one instance of the game should exist in memory at any given time.** This means strict control over how game sessions are created and accessed must be implemented. The Singleton pattern will be essential in enforcing this constraint, ensuring that all interactions reference the same instance.
2. **Game and team names must be unique.** A centralized name registry should be implemented, backed by a database or in-memory cache. Developers will need to build logic that checks for name uniqueness in real time and handles name conflicts.
3. **Each game must support multiple teams, and each team must support multiple players.** Object-oriented principles and well-defined relationships will help to manage associations between game objects (game, teams, players) cleanly.
4. **The application must be accessible on multiple platforms.** Front and back-end development languages should be compatible cross-platform, and developers must design with multi-platform compatibility in mind.

## [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 below UML class diagram outlines an object-oriented design for development of *Draw It or Lose It*. The Entity superclass provides shared attributes id and name to its subclasses: Game, Team, and Player. Each Game can have multiple Team objects, and each Team can have multiple Player objects, reflecting the game’s multiplayer and team-based structure. The use of inheritance supports code reusability and simplifies management of common data among classes.

A key feature of the design is the GameService class, which implements the Singleton pattern to ensure that only one instance of the game controller exists in memory at any given time. This class manages all game-related operations, such as generation of unique identifiers for games, teams, and players. The diagram also shows principles such as encapsulation, composition, and abstraction, which contribute to an efficient and scalable design that aligns with The Gaming Room’s requirements for building a web-based, muti-platform game application.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS can support local development and testing environments for web-based apps, but it’s rarely used in production due to hardware restrictions, cost, and limited server tooling. | Linux is the industry standard for web app servers due to its open-source nature, low cost, high performance, and stability. It supports common web stacks like LAMP/LEMP and is ideal for scalable applications. | Windows Server supports enterprise-level applications and integrates well with Microsoft technologies. However, it comes with licensing costs and is less compatible with many open-source tools. | Mobile devices are not suitable for server-side hosting. They lack persistent network connections, power, and administrative access. They are best treated as clients in a web-based application model. |
| **Client Side** | macOS ensures compatibility with Apple devices and browsers like Safari, making it valuable for testing. However, the development requires Apple hardware, which adds cost. | Linux is a low-cost, flexible OS with excellent tools for development and testing. While it lacks broad desktop market share, it is useful for web dev and supports all major browsers. | Windows has broad usage and supports most development tools and browsers. Ensuring support across various versions may add complexity but helps reach a wide audience. | Supporting iOS and Android requires either native development (Xcode for iOS, Android Studio) or cross-platform tools like Flutter or React Native. Cross-platform approaches reduce long-term dev time. |
| **Development Tools** | Tools like Xcode (for iOS), Visual Studio Code, and Terminal make macOS effective for full-stack development and necessary for iOS deployment. | Linux supports IDEs such as Eclipse, IntelliJ, and VS Code, and works well with web stacks (JavaScript, Python, PHP). Ideal for open-source development workflows. | Windows supports Visual Studio, PowerShell, and Windows Subsystem for Linux (WSL), making it versatile. It's ideal for .NET and enterprise tools, but licensing may be a factor. | Mobile development benefits from platform-specific IDEs (Xcode, Android Studio) or shared frameworks (Flutter, React Native, Unity). These enable broader reach without duplicating codebases. |

## 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**: A Linux-based server is the best fit for Draw It or Lose It due to its low cost, stability, and broad support for scalable, modern web applications. For the client side, the application should be web-based and responsive, ensuring compatibility across desktop (Windows/macOS/Linux) and mobile (iOS/Android) browsers. Using JavaScript frameworks like React or Vue.js supports a consistent UI across platforms while reducing duplication of development effort. This stack supports CI/CD automation, automated testing, and rapid deployment workflows.
2. **Operating Systems Architectures**: Linux’s modular and monolithic architecture allows fine-grained control over system resources, process scheduling, and memory management. It integrates seamlessly with container-based deployments, such as Docker, enabling resource-efficient, isolated environments for game instances.
3. **Storage Management**: Cloud-based storage solutions such as Amazon S3, Google Cloud Storage, or Azure Blob Storage are optimal for handling game data, assets (e.g., stock images), and user profiles. These services provide redundancy, fast delivery, and simple APIs for secure access, ensuring the game scales with user demand.
4. **Memory Management**: Linux excels in memory optimization via techniques like virtual memory, swapping, and cache management. These tools allow dynamic allocation for different parts of the game (game state, player sessions, image buffers), ensuring responsive performance without overconsumption of server resources.
5. **Distributed Systems and Networks**: The game should use RESTful APIs for standard data interactions and WebSockets for real-time gameplay elements (e.g., drawing progression, timers, and guesses). Hosting this infrastructure in the cloud (e.g., Azure App Services or AWS EC2 with Elastic Load Balancing) allows the application to scale and recover quickly from failures or outages. Client devices will interact with stateless services, improving scalability and reliability under high traffic.
6. **Security**: To meet privacy and security requirements, HTTPs should be enforced site-wide to protect data in transit, OAuth2 or similar token-based authentication should be used for login, role-based access controls should prevent unauthorized changes or access to game states, data encryption at rest and secure cloud storage permissions must be enforced, and regular security audits, firewall configuration, and updates/patches should be automated via a CI/CD pipeline. This multi-layered approach ensures that user data is protected at every stage, from login to gameplay and data storage.