

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 | 09/18/25 | Courtney McDonald | All contents of the template has been filled. |

**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 currently offers *Draw It or Lose It* as an Android app and is looking to expand it into a **web-based platform** that can be accessed on multiple devices. The game involves teams competing to guess drawings rendered from a stock image library, with gameplay divided into timed rounds.

At this time, the client’s staff does not have the technical expertise to set up and manage a web-based environment, so a reliable system is needed to handle **games, teams, and players**, enforce gameplay rules, and prevent duplicate entries or errors.

The proposed solution is a **web-based gaming platform** designed to manage all aspects of gameplay while ensuring scalability and maintainability. Key features include:

* **Singleton Game Instance:** Ensures that only one game is active in memory at a time, preventing conflicts.
* **Team and Player Management:** Supports multiple teams and players, with automatic validation to ensure unique names.
* **Gameplay Mechanics:** Four one-minute rounds per game, with images rendered steadily and fully by the 30-second mark, and opportunities for extra guesses if necessary.
* **Scalable and Extensible Design:** Built using object-oriented principles to support future enhancements such as leaderboards or game analytics.
* **Cross-Platform Accessibility:** Accessible on desktops, tablets, and mobile browsers.

**Critical Client Information:**

* All names (game, team, player) must be unique; the system will automatically enforce this.
* The system enforces timing rules and game mechanics to maintain fair play.
* CTS will provide guidance to set up the development environment and support the web-based deployment.

## Requirements

**Business Requirements:**

* The game must accommodate multiple teams competing simultaneously.
* Teams must be able to have multiple players assigned.
* Only one active game instance should exist at a time to avoid conflicts.
* The game should be accessible on multiple platforms (web, desktop, tablet).

**Technical Requirements:**

* Unique identifiers for games, teams, and players.
* Automatic validation to prevent duplicate names.
* Proper enforcement of gameplay timing and rules.
* Scalable and extensible architecture for future enhancements.
* Web-based platform that supports distributed access.

## [Design Constraints](#_2et92p0)

## The development of the web-based *Draw It or Lose It* game must take several design constraints into account to ensure the system is reliable, scalable, and functional across multiple platforms, I included both the constraint and reasoning/consideration:

**Web-Based Distributed Environment**

* The application must support access from multiple clients over the internet, including desktops, tablets, and mobile browsers.
* Considerations: The system requires proper session management, network communication handling, and synchronization of game state across multiple clients to maintain consistency.

**Singleton Game Instance**

* Only one active game instance can exist in memory at any time.
* Considerations: Prevents conflicts between simultaneous games, but requires careful design to manage game state and ensure the system can handle multiple users without errors.

**Unique Identifiers for Data Integrity**

* Each game, team, and player must have a unique identifier, and duplicate names must be prevented.
* Considerations: Validation logic must be implemented to enforce uniqueness, which ensures accurate tracking of players, teams, and games, and prevents data inconsistencies.

**Scalability and Extensibility**

* The system must be designed to allow future features such as leaderboards, game analytics, or additional game modes.
* Considerations: The architecture must follow object-oriented principles and modular design to support future growth without major redesigns.

## [System Architecture View](#_ilbxbyevv6b6)

The system can be logically viewed as:

* **Client Tier:** Web browser or mobile browser interface where players interact with the game.
* **Application Tier:** Server-side logic manages game instances, teams, players, and gameplay rules.
* **Data Storage Tier:** Database stores game, team, and player information with unique identifiers, ensuring data integrity.
* **Communication:** Clients communicate with the server over HTTP/HTTPS; the server manages session state and ensures synchronization between players.

## [Domain Model](#_8h2ehzxfam4o)

This UML diagram outlines the Gaming Room application and how its classes work together. The Entity class is the base for Game, Team, and Player, giving them shared attributes like id and name. This shows inheritance and avoids repeating code.

The GameService class uses the singleton pattern, ensuring only one instance controls all games, teams, and players. It manages lists of games, assigns unique IDs, and prevents duplicate names when new teams or players are added.

Relationships are clear: a Game holds multiple Teams, and each Team holds multiple Players. This demonstrates aggregation, modeling how these objects depend on each other.

Other principles are also present: encapsulation (private attributes with getters/setters) and composition (teams inside games, players inside teams). The ProgramDriver runs the system, while SingletonTester confirms the singleton behavior works as expected.

Altogether, the design uses OOP principles to efficiently manage data integrity, uniqueness, and organization across the 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** | Mac can host web applications using Apache, Nginx, or Node.js. It is a stable and reliable environment for development, but high hardware costs and limited production adoption may constrain scalability. | Linux is highly recommended for server deployment. It is secure, robust, and optimized for high-performance workloads. Its flexibility and scalability make it ideal for hosting a web-based, multi-user game. However, it requires technical proficiency with command-line tools and server administration. | Windows Server offers simple setup and excellent integration with Microsoft technologies. Licensing fees are higher, and additional attention is needed for security hardening. Performance is reliable but generally less efficient that Linux under heavy load. | Mobile devices aren’t really suited to be servers. They can only host local simulations or testing environments due to limited processing power, storage, and network capabilities. |
| **Client Side** | Mac desktops support all modern browsers. Testing across multiple versions is required. Hardware costs are high but reliability and developer tools are strong. | Linux desktops support modern browsers but have lower market share, requiring more rigorous testing across distributions and browser configurations. | Windows desktops are widespread and compatible with all major browsers, Version differences and licensing are considerations, but support and troubleshooting are generally easier. | Mobile clients (iOS/Android) necessitate responsive design, touch interface optimization, and careful performance tuning. Development may require cross-platform frameworks or native apps to ensure functionality across devices. |
| **Development Tools** | Supports Java, Python, JavaScript; Node.js, and web development frameworks. IDEs like Xcode, IntelliJ, VS Code are widely used. Strong Unix tools help with cross-platform development and automation testing. Mac hardware is expensive but most development tools are free and no server OS license is needed for the development environments. | Supports almost all languages; IDEs include Eclipse, VS Code, VS Code, and IntelliJ. Command-line tools and package managers make automation and deployment easier.Linux is mostly free with the exception of enterprise distributions like Red Hat requiring subscription fees. IDEs and development tools are usually free, | Supports Java, Python, C#, JavaScript; IDEs like Visual Studio, IntelliJ, VS Code. Works well for Windows clients but some open-source setup may be trickier. Windows Server licenses can be more. Some enterprise IDEs can require paid licenses. | Mobile dev uses Swift (iOS), Kotlin/Java (Android), or cross-platform frameworks like React Native/Flutter. IDEs are Xcode and Android Studio; testing across devices is key. Xcode is free and Apple Developer Program is $99/year while Android studio is free. Deployment for android has no extra licensing 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**: For hosting Draw It or Lose It, I recommend Linux as the primary server platform. Linux offers unmatched stability, high performance, and robust security, making it ideal for a web-based gaming environment that is required to scale to thousands of users. Its open source nature reduces licensing costs, and it is widely adopted in distributed and cloud-based systems. Linux servers also support containerization and orchestration tools like Docker and Kubernetes, which allow seamless deployment and management of multiple game instances.
2. **Operating Systems Architectures:** The Linux server will follow a multi-tier architecture to support scalability and maintainability. Presentation tier, web servers handle HTTP requests from browsers and mobile clients, delivering responsive HTML interfaces.Application tier, stateless application servers manage game logic, session handling, timers, and scoring. Multiple instances can run concurrently, leveraging Linux process isolation.Data Tier, where databases store persistent game, team, and player information. Redis caches game states for real-time gameplay and distributed locking mechanisms. This architecture ensures efficient resource utilization, and fault tolerance, allowing the game to operate reliably and accurately across various client platforms.
3. **Storage Management**: A hybrid storage system is recommended with some forms as Relational Database (PostgreSQL), which stores structured game data, including users, teams, and game history, with strong data integrity and transaction support.In-Memory Cache (Redis), which temporarily stores active game states, session data, and distributed locks to enable real-time updates without database bottlenecks.Object Storage (S3-Compatible Storage), which stores high-definition images for rendering, enabling scalable and globally accessible content delivery.  
   This combination optimizes read/write performance, minimizes latency, and ensures that high-volume data can be efficiently managed and accessed across platforms.
4. **Memory Management:** Linux employs several memory management techniques essential for *Draw It or Lose It* such as **Virtual Memory:** Extends physical RAM using disk space to support multiple high-memory processes, such as rendering 8 MB images across several concurrent game instances. **Paging and Swapping:** Allows large images to be loaded on demand without exhausting physical memory. **Memory Pooling and Buffering:** Reduces allocation overhead during frequent image rendering operations. **Threaded Execution:** Enables rendering tasks to execute concurrently while the main game loop remains responsive. These approaches ensure smooth, low-latency gameplay even under high user loads.
5. **Distributed Systems and Networks:** To support cross-platform communication, the system will utilize a distributed architecturesuch asRESTful APIs that enable web clients on desktops and mobile devices to interact with the server in a platform-agnostic manner.WebSockets provide real-time communication for gameplay updates, ensuring teams receive images and guesses without noticeable lag. Load Balancing distributes traffic across multiple servers to prevent single points of failure. Redundancy and Failover which allows critical game state data to be replicated across nodes and Redis clusters to maintain continuity during outages. This distributed approach ensures that *Draw It or Lose It* can scale horizontally and maintain consistent gameplay experiences across devices.
6. **Security:** Securing user information is imperative. Some measures to consider are encrypted communication which allows all traffic between clients and servers will use HTTPS.Authentication and Authorization has role-based access controls and token-based authentication (JWT) prevent unauthorized access. Authentication and authorization also has role-based access controls and token-based authentication (JWT) prevent unauthorized access.Operational security would involve regular patching, monitoring, and secure secret management to protect against vulnerabilities and ensure regulatory compliance. These measures ensure that both players and administrators can interact with the game securely across all platforms.