

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 | 01/22/25 | Ethan Chapman | First Iteration |
| 2.0 | 02/05/2025 | Ethan Chapman | Second Iteration |
| 3.0 | 2/9/2025 | Ethan Chapman | Third Iteration |
| 4.0 | 2/21/2025 | Ethan Chapman | Fourth Iteration |

**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 "Draw It or Lose It" game is a web-based, cross-platform application designed to provide an engaging multiplayer experience. Unlike traditional drawing games, this application uses a pre-existing library of images to generate clues, ensuring consistency and fairness. The game consists of four rounds, each lasting one minute, with drawings rendered gradually and completed at the 30-second mark. If a team fails to guess the correct answer before time expires, other teams have a 15-second opportunity to make one guess each.

To expand accessibility, the game will support multiple platforms, including Linux, Mac, Windows, and mobile devices. The Gaming Room requires an evaluation of the characteristics, advantages, and weaknesses of these platforms to determine the best approach for deployment. This document provides a comparative analysis of operating platforms, development tools, and security considerations, ensuring an optimized distributed system capable of handling thousands of concurrent players.

## Requirements

* **Multiple Teams:** The game should support one or more teams competing simultaneously.
* **Multiple Players per Team:** Each team can consist of multiple players, fostering cooperative gameplay.
* **Unique Naming:** Game and team names must be unique to prevent duplication and confusion.
* **Singleton Instance:** A single instance of the game should be maintained in memory at any given time to optimize resource usage.
* **Web-Based Deployment:** The game must be hosted on a scalable web server, ensuring high availability and responsiveness across multiple platforms.
* **Cross-Platform Compatibility:** The game should be accessible via desktop browsers (Linux, Mac, Windows) and mobile devices (iOS and Android), using a responsive HTML interface.
* **Performance Efficiency:** Drawings must be fully rendered within 30 seconds, with smooth real-time interactions for players.
* **Fair Competition:** If a team does not guess correctly within the allocated time, other teams must have an opportunity to submit a single guess within 15 seconds.
* **Security Measures:** The application must implement authentication, encryption, and secure communication protocols to protect user data.

## [Design Constraints](#_2et92p0)

Developing the game as a web-based distributed application presents several constraints:

1. **Scalability:** The system must handle multiple concurrent players and teams efficiently.
2. **Data Integrity:** Unique game, team, and player names must be maintained without duplication.
3. **Concurrency Control:** Only one instance of the game should exist in memory at any given time, necessitating synchronization mechanisms.
4. **Latency:** The drawing render process must be optimized to ensure smooth gameplay within the 60-second time frame.
5. **Security:** User data must be protected through authentication, encryption, and secure communication protocols.
6. **Cross-Platform Compatibility:** The application must run seamlessly on various operating systems and devices.

These constraints influence the application development by necessitating efficient data storage, cloud-based deployment, and robust backend processing.

## [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 "Draw It or Lose It" game represents the core entities and their relationships:

1. **Entity Class:** A base class for all objects, encapsulating common attributes such as unique identifiers.
2. **Game Class:** Manages the overall game logic, with relationships to teams and players.
3. **Team Class:** Contains multiple players and has a unique identifier to ensure distinct team names.
4. **Player Class:** Represents individual participants within teams.

Object-oriented principles demonstrated include:

* **Encapsulation:** Each class contains attributes and behaviors relevant to its scope.
* **Inheritance:** The Entity class provides a shared structure for all entities.
* **Polymorphism:** Future extensions can override base behaviors for different game modes.

This design efficiently meets the software requirements by ensuring uniqueness, proper team structuring, and streamlined game management.

**"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** | Mac provides a UNIX-based environment with good security but lacks widespread server adoption compared to Linux. Hosting costs may be higher due to limited support for enterprise solutions. | Linux offers high scalability, security, and cost-effective deployment. It supports cloud services like AWS, Google Cloud, and Azure, making it an ideal choice for web-based applications. | Windows Server provides strong enterprise support but comes with significant licensing costs. It integrates well with Microsoft technologies but is less optimized for high-performance web hosting. | Mobile devices are not suitable for hosting a web-based server application due to hardware and resource limitations. |
| **Client Side** | Developing for macOS requires using Swift and Xcode. While macOS provides a high-performance environment, it has a smaller market share compared to Windows. | Linux offers flexibility, but cross-platform compatibility issues may arise. Web-based deployment ensures better accessibility. | Windows has the largest desktop user base, making it critical for application compatibility. Development can use .NET, Java, and Electron to support multiple platforms. | Mobile development requires supporting both iOS and Android, necessitating cross-platform frameworks such as React Native or Flutter to reduce development time. |
| **Development Tools** | Requires Xcode, Swift, and additional web technologies like React and Node.js. Development can be costly due to Apple's restrictions. | Supports multiple languages (Python, Java, Node.js) with free development tools. Ideal for web-based applications. | Uses Visual Studio, .NET, and other proprietary tools, which may include licensing fees. Strong compatibility with enterprise applications. | Requires Android Studio for Android and Xcode for iOS. Cross-platform frameworks like Flutter and React Native can help streamline development. |
| **Licensing Costs** | macOS Server has a one-time purchase cost but is not as widely supported for large-scale hosting. | Linux is open-source and free, making it the most cost-effective option for server deployment. | Windows Server requires a licensing fee, which increases overall costs. Additional licenses for enterprise features may be necessary. | Mobile platforms do not incur server-side licensing costs since they do not host the application. |

**Recommendations:**

1. Operating Platform

Recommendation: The best operating (server) platform for The Gaming Room to expand Draw It or Lose It across multiple computing environments is Linux (Ubuntu Server).

Justification:

- Cross-platform compatibility: Linux servers are widely supported by Android, iOS, Windows, and web applications.

- Scalability: Ubuntu Server offers robust support for cloud environments (AWS, Azure, Google Cloud) for future growth.

- Cost-effective: Open-source with no licensing fees.

- Community and Support: Extensive documentation and community support.

2. Operating Systems Architectures

Chosen Architecture: Microkernel Architecture (Linux-based)

Details:

- Modular structure: The microkernel handles only essential functions like memory management and inter-process communication, while additional services run in user space, improving flexibility and fault isolation.

- Portability: Easier to adapt the game to other operating systems (Windows, macOS) as only user-space services need adjustment.

- Security: Minimizes vulnerabilities by running minimal services in kernel mode.

3. Storage Management

Recommendation: Network File System (NFS) with a NoSQL database (MongoDB)

Justification:

- NFS: Ensures efficient sharing of game data and assets across multiple servers and platforms.

- MongoDB: Offers high availability, scalability, and performance for storing unstructured data like user profiles and game progress.

- Backup & Recovery: Cloud-based storage solutions (e.g., AWS S3) for redundancy and quick recovery from outages.

4. Memory Management

Memory Management Techniques on Linux (Ubuntu Server):

- Paging: Enables efficient allocation of memory to different processes by loading only required memory pages.

- Swapping: Moves inactive memory pages to disk storage, optimizing RAM usage during high-demand periods.

- Buffer and Cache Management: Speeds up read/write operations by caching frequently accessed data.

- Garbage Collection: For applications running in JavaScript (Node.js backend), automatic garbage collection optimizes memory utilization.

Impact on "Draw It or Lose It": These techniques ensure smooth real-time performance and responsiveness by efficiently managing server resources, even during peak gameplay.

5. Distributed Systems and Networks

Recommendation: Client-Server Distributed Architecture with RESTful APIs and WebSocket Protocol

Key Components:

- RESTful APIs: For secure communication between client applications and the central server for routine data retrieval and storage.

- WebSockets: For real-time data exchange (gameplay interactions) to maintain low-latency performance across devices.

- Load Balancers: Distribute network traffic evenly across multiple server instances, reducing downtime risks.

- Content Delivery Network (CDN): Enhances load times and ensures high availability globally.

Dependencies Management:

- Connectivity: Failover systems to reroute traffic during server outages.

- Scalability: Cloud services (AWS Elastic Beanstalk) to scale server capacity based on demand.

- Monitoring: Continuous network monitoring tools (e.g., Prometheus, Grafana) to detect and resolve issues proactively.

6. Security

Key Security Measures:

- Data Encryption: End-to-end encryption (TLS/SSL) for data transmitted between user devices and servers.

- Authentication: OAuth 2.0 protocol for secure user authentication and authorization.

- Secure APIs: Input validation and rate limiting on all API endpoints to prevent injection attacks and DDoS attempts.

- Data Privacy: Adherence to GDPR and CCPA regulations by anonymizing user data and implementing user consent protocols.

- Network Security: Firewalls and intrusion detection systems (IDS) to prevent unauthorized access and detect malicious activities.

- Regular Updates: Frequent security patching for the OS and all application dependencies.

Conclusion:

By adopting Ubuntu Server (Linux) with microkernel architecture, NFS and MongoDB for storage, efficient memory management techniques, and a client-server distributed system supported by WebSockets and RESTful APIs, The Gaming Room can seamlessly expand Draw It or Lose It to multiple platforms. The proposed security framework ensures robust protection of user data and system integrity, aligning with the client’s expansion and user protection goals.