

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 | 11/17/2024 | Jose Moreno | Initial draft of the software design document. First developed and tested version. |
| 1.2 | 12/02/2024 | Jose Moreno | Secondary software design document. Development progress, and previous testing included. |
| 1.3 | 12/15/2024 | Jose Moreno | Final software design document. Including final recommendations to successfully complete software. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has tasked Creative Technology Solutions (CTS) with developing a web-based version of their popular Android game, *Draw It or Lose It*. The objective is to create a scalable, cross-platform application accessible via multiple devices while maintaining the core game mechanics and unique features of the original version.

This document outlines the design strategy, requirements, constraints, and recommendations for achieving a streamlined development process for the web-based version. The aim is to ensure a robust, secure, and user-friendly application while adhering to the client’s specifications.

## Requirements

*The following software requirements are essential for the new web-based version of Draw It or Lose It:*

1. ***Game Structure****:*
   * *Support for one or more teams per game.*
   * *Each team can have multiple players assigned.*
2. ***Name Uniqueness****:*
   * *Game and team names must be unique, enabling real-time validation to prevent duplicates.*
3. ***Instance Management****:*
   * *Ensure only one instance of the game exists in memory at a time.*
   * *Assign unique identifiers to games, teams, and players to maintain integrity and prevent conflicts.*

## [Design Constraints](#_2et92p0)

Web-Based Environment: The application must be designed for a web-based distributed environment, accessible on various platforms (desktops, tablets, and mobile devices).

Implications: Requires responsive design, compatibility with different operating systems, and efficient network communication.

Scalability: The game should accommodate multiple users simultaneously without performance degradation.

Implications: Scalable backend architecture and database design are crucial.

Security: User data (e.g., team names, scores) must be protected using robust security measures.

Implications: Encrypt communication, implement secure user authentication, and protect against common vulnerabilities like SQL injection.

## [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)

Classes:

Game: Contains attributes for unique ID, game name, teams, and status.

Team: Contains attributes for unique ID, team name, players, and score.

Player: Contains attributes for unique ID, player name, and team association.

Relationships:

A Game contains multiple Team objects.

Each Team contains multiple Player objects.

Objects are linked via unique identifiers to maintain integrity.

**"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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Reliable performance and strong development tools. Limited customization.  Server support available but not as common for large-scale deployments.  High hardware and software costs.  Advantages: Stable and integrates well with Apple’s ecosystem.  Disadvantages: Limited scalability and fewer server-focused tools compared to Linux and Windows. | Highly customizable, cost-effective, and performant. Ideal for servers.  Robust support for server hosting with tools like Apache, Nginx, and Docker. Ideal for scalable web apps.  Open-source; negligible costs.  Advantages: Highly customizable, reliable performance, and extensive community support.  Disadvantages: Requires skilled developers for optimal setup and maintenance. | Broad compatibility but higher licensing costs.  Strong server support with IIS and Azure integration.  Higher licensing costs for Windows Server.  Familiarity for many developers and enterprise-grade support.  Costly, with resource-heavy requirements compared to Linux. | Limited for hosting. Best suited for client-side features.  Device-specific costs (Ex: App Store/Play Store fees).  Advantages: Excellent client-side performance for apps  Disadvantages: Unsuitable for hosting a scalable web application. |
| **Client Side** | Excellent UI development support but higher hardware costs.  Strong browser support (Safari, Chrome, Firefox).  Higher development and hardware costs.  Advantages: Excellent for UI design and development.  Disadvantages: Expensive for small teams or startups. | Lightweight and flexible. Requires experienced developers.  Compatible with all major browsers.  Requires testing across distributions to ensure consistency.  Advantages: Lightweight and flexible for web development.  Disadvantages: Limited market share for desktop clients. | Familiar environment but higher resource demand.  Compatible with all major browsers.  Familiarity with development tools like Visual Studio can simplify development.  Advantages: Widely used, making it a primary target for desktop applications.  Disadvantages: Higher hardware and licensing costs. | Essential for client use with responsive design.  Native app compatibility for iOS and Android.  Requires responsive design and tools like React Native or Flutter.  Advantages: Expands reach to a broader audience.  Disadvantages: Fragmentation of Android devices can complicate testing. |
| **Development Tools** | Xcode, VS Code, Node.js | Eclipse, VS Code, Node.js | Visual Studio, Node.js, Java, Eclipse | Android Studio, React Native, Xcode (iOS) |

**Development Tools**

* **Licensing Costs:**
  + **Open-source tools (Linux, VS Code, Node.js) keep costs low.**
  + **Paid tools (Windows Server, Visual Studio Pro, or Mac hardware for Xcode) add to expenses.**
* **Team Impact: Supporting multiple platforms may necessitate specialized teams (e.g., iOS developers for Xcode, Android developers for Android Studio).**

**Let me know if you'd like to refine any specific part of this evaluation or discuss additional recommendations.**

## Recommendations

## Operating Platform

## To support the expansion of *Draw It or Lose It* across various computing environments, I recommend using a cloud-based platforms. These platforms provide scalable server environments that support multiple operating systems, including Windows, Linux, and macOS.

## Reasons:

## Cloud platforms allow dynamic scaling to handle increased user loads.

## Cross-platform support because it enables smooth integration with various operating systems and devices.

## Cost efficiency since pay-as-you-go models reduce upfront costs.

## Global availability because data centers worldwide ensure low latency and high availability.

## Operating Systems Architectures

## The recommended platforms support the following architecture:

## Linux: A lightweight, open-source architecture with robust security and high customizability.

## Windows Server: A familiar architecture for .NET or Windows-based applications with strong integration tools.

## macOS: Less commonly used for servers but compatible with specific applications requiring Apple’s ecosystem.

## Storage Management

## The recommended storage management solution is AWS Elastic File System (EFS) or Azure Blob Storage. These solutions provide:

## Automatic scaling with demand.

## File sharing across Linux, Windows, and macOS environments.

## High durability and availability as data is replicated across multiple locations to ensure reliability.

## Easily integrates with distributed applications to manage persistent data storage.

## Memory Management

## Modern operating systems utilize several memory management techniques, including:

## Virtual Memory which is used to optimize RAM usage by storing inactive data in swap space.

## Paging and segmentation used to divide processes into smaller chunks for efficient memory utilization.

## Garbage Collection which automatically cleans up unused memory to prevent memory leaks.

## Dynamic Allocation which allocates memory as needed, reducing overhead.

## For *Draw It or Lose It*, the chosen platform’s memory management capabilities ensure:

## Efficient handling of simultaneous user interactions.

## Low latency for real-time game interactions.

## Proper allocation of resources to maintain performance during high traffic.

## Distributed Systems and Networks

## *Draw It or Lose It* will leverage distributed systems and networks to communicate seamlessly between platforms. This will be achieved through:

## Distributed Software where the game logic and databases will use APIs or microservices to enable platform-independent communication.

## Networking by using of protocols such as HTTP/HTTPS for secure web communication and WebSocket for real-time updates.

## Load Balancers that distribute user requests evenly across servers, ensuring reliability.

## Redundancy for cloud-based backups and failovers to mitigate downtime caused by outages.

## Security

## To ensure robust user protection, the following measures will be implemented:

## Data Encryption which use TLS/SSL for data transmission for stored data.

## Authentication and Authorization.

## Regular Updates: Apply security patches promptly to address vulnerabilities in software.

## Multi-factor Authentication which adds an extra layer of user protection.

## Data Anonymization to minimize exposure of sensitive user information.

## In conclusion, by adopting these recommendations, The Gaming Room can successfully expand *Draw It or Lose It* into a distributed, secure, and scalable multi-platform environment.