

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 | 07/18/2024 | | Jean Pierre | | | Initial draft of the software design document for the web-based game application "Draw It or Lose It. |
| 2.0 | | 08/01/2024 | | Jean Pierre | Updated information related to the software design. | | |
| 3.0 | | 08/15/2024 | | Jean Pierre | Architecture Recommendation | | |

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)

"Draw It or Lose It" is a web-based game application inspired by the 1980s television game "Win, Lose or Draw." The game involves teams competing to guess what is being drawn. Unlike the original, where players draw images, our application will use a library of stock drawings rendered progressively over a minute. This document outlines the software design, addressing critical client requirements to ensure a streamlined development process. Essential requirements include supporting multiple teams, unique games and team names, and managing a single game instance in memory.

## Requirements

The client has outlined the following business and technical requirements for the game application:

- The game must support multiple teams, each with several players.

- Unique names for games and teams to prevent duplication and confusion.

- A single instance of the game in memory at any given time, managed by unique identifiers for games, teams, and players.

## [Design Constraints](#_2et92p0)

Developing "Draw It or Lose It" in a web-based distributed environment involves several design constraints:

1. Scalability: The application must handle numerous concurrent games and users efficiently.

2. Unique Identifiers: Implementing unique identifiers for games, teams, and players is crucial to prevent conflicts and maintain data integrity.

3. Concurrency Management: Ensuring only one game instance exists in memory simultaneously requires robust concurrency control.

4. Real-Time Rendering: Smooth gameplay requires efficient real-time rendering of images across various devices and browsers.

Implications:

Database Design: Must support efficient storage and retrieval of unique identifiers and game states.

Server Performance: High-performance servers and optimized code are essential for real-time rendering and multi-user interactions.

Security: Strong security measures are needed to protect user data and maintain application integrity.

## [System Architecture View](#_ilbxbyevv6b6)

While not required for this project, it is crucial to consider the overall system and subsystem architecture, including physical components and communication topology, for future reference.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram for "Draw It or Lose It" includes the following classes:

Game: Manages the game state with a unique identifier, including teams and rounds.

Team: Represents a team containing multiple players and a unique team identifier.

Player: Represents an individual player, with attributes such as name and unique player identifier.

Entity: A base class holding common attributes and behaviors shared by Game, Team, and Player classes.

Class Relationships and OOP Principles:

Inheritance: The Entity class promotes code reuse and encapsulation by sharing common attributes across Game, Team, and Player classes.

Association: The Game class is associated with multiple Team objects, and each Team object is related to various Player objects, reflecting real-world relationships.

Encapsulation: Each class encapsulates its data and behaviors, ensuring a modular and maintainable codebase.

Polymorphism: Methods in the Entity class can be overridden by subclasses, enhancing flexibility and extensibility.

"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 | characteristics: Stable and user-friendly. Advantages: Robust tools and seamless Apple integration. Weaknesses: Higher costs, less flexible. Deployment: Supports macOS Server. Licensing Costs: High. | characteristics: Highly flexible, cost-effective. Advantages: Open-source support, strong community. Weaknesses: Requires technical expertise. Deployment: Supports LAMP stack. Licensing Costs: Low to none. | characteristics: Wide compatibility. Advantages: Extensive support and a large user base. Weaknesses: Higher licensing costs. Deployment: Supports Windows Server and IIS. Licensing Costs: Potentially high. | characteristics: High portability and accessibility. Advantages: Ease of access for users. Weaknesses: Limited resources. Deployment: Limited to back-end support. Licensing Costs: Varies. |
| Client Side | considerations: Costly, seamless iOS integration. Cost: High. Time: Longer, requires Swift/Objective-C expertise. Compatibility: Seamless with iOS. | considerations: Cost-effective, versatile. Cost: Lower. Time: Varies, requires open-source tool expertise. Compatibility: Supports various open-source languages. | considerations: Extensive support and robust tools. Cost: High. Time: Moderate familiarity with Visual Studio/.NET needed. Compatibility: Strong for Windows applications. | considerations: Different OS environments. Cost: Varies. Time: This can be longer due to optimization compatibility: Modern, responsive HTML interfaces. |
| Development Tools | Tools, Tools: XCode, Xamarin, Flutter. Languages: HTML, CSS, JavaScript, Swift, Objective-C. Impact: Higher costs, specialized team. | tools: Eclipse, NetBeans, VS Code. Languages: HTML, CSS, JavaScript, Python, PHP, Ruby. Impact: Versatile skill set required. | tools: Visual Studio, .NET. Languages: HTML, CSS, JavaScript, C#, NET. Impact: Comprehensive environment, higher%20costs. | Tools: Android Studio, XCode, Flutter, React Native. Languages: HTML, CSS, JavaScript, Kotlin, Swift. Impact: Knowledge of multiple environments is needed. |

## Recommendations

Analyze the characteristics and techniques of various systems architectures and recommend The Gaming Room. Specifically, address the following:

1. Operating Platform Recommendation

Given the requirement to expand "Draw It or Lose It" across various computing environments, I recommend using a cross-platform solution built on Linux servers, utilizing Docker containers. Linux provides a robust, secure, and scalable environment with extensive support for various software technologies and platforms. Docker allows packaging the game and its dependencies into a container, ensuring consistent performance across different operating systems (e.g., Windows, macOS, Linux).

2. Operating Systems Architectures

The recommended Linux platform operates on a monolithic kernel architecture. This architecture integrates the operating system's core functions—such as file system management, process scheduling, and device drivers—into a single kernel. This approach offers high performance and efficiency, crucial for managing the game's demands. Additionally, Docker's architecture leverages the Linux kernel's capabilities, including groups (for resource allocation) and namespaces (for isolating processes), to ensure that each container operates independently.

3. Storage Management

I recommend using ZFS (Zettabyte File System) on the Linux server for storage management. ZFS provides advanced features like data compression, snapshots, and data integrity checks, which are beneficial for maintaining large datasets like those required for a gaming environment. ZFS also simplifies storage management by allowing dynamic resource allocation and supporting high performance and data security.

4. Memory Management

The Linux platform offers sophisticated memory management techniques, including paging and swapping. The kernel efficiently manages physical memory through paging, which ensures that the system uses memory efficiently by only loading necessary program parts into RAM. If the system runs low on RAM, swapping comes into play, where less-used, data is temporarily moved to disk storage (swap space). Additionally, Docker containers are isolated from each other in terms of memory usage, meaning each container gets its allocated memory resources, preventing one container from exhausting the system's memory.

5. Distributed Systems and Networks

To enable "Draw It or Lose It" to communicate between various platforms, I recommend adopting a microservices architecture deployed across a distributed environment using Kubernetes. Kubernetes orchestrates Docker containers across multiple nodes (servers), allowing for scalable, resilient, and distributed deployment of services. Service communication can be managed through RESTful APIs or gRPC, ensuring interoperability between different systems. The network that connects these devices should rely on Software-Defined Networking (SDN) for flexible and efficient traffic management. Furthermore, load balancers ensure that requests are evenly distributed across servers, improving fault tolerance and performance.

6. Security

Security is paramount, especially in a distributed environment. The recommended Linux platform, along with Docker and Kubernetes, provides several layers of security:

User and process isolation: Docker containers run processes in isolation, preventing unauthorized access to the host system and other containers.

TLS/SSL encryption: Implement end-to-end encryption for all data in transit between client devices and the server, ensuring that sensitive user information is protected.

Access control and authentication: Implement robust authentication mechanisms such as OAuth2 for user logins and API access, ensuring only authorized users can access the game.

Data encryption at rest: Use ZFS encryption features to protect user data stored on disk, ensuring the data remains secure even if the physical storage is compromised.

Regular security audits and updates:

Continuously monitor for security vulnerabilities and apply updates promptly. Utilize automated tools to scan for vulnerabilities in the containers and underlying OS.

Conclusion

By leveraging a Linux-based platform with Docker and Kubernetes, The Gaming Room can expand "Draw It or Lose It" across multiple platforms efficiently and securely. The proposed architecture provides scalability, performance, and security to support the game's growth while maintaining a consistent user experience across different environments.