

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 | 09/22/2024 | Keyvan Gharehtapeh | Exec Summary and Design Constraints |
| 2.0 |  |  |  |
| 3.0 | 10/14/2024 | Keyvan Gharehtapeh |  |

## [Executive Summary](#_sbfa50wo7nsh)

This document outlines the proposed software design for The Gaming Room's web-based game, inspired by their existing Android app, "Draw It or Lose It." The game will be adapted to function seamlessly across multiple platforms, offering a competitive and engaging experience where teams race to guess images rendered from a vast stock library.

## Requirements

Key features of the game include:

* **Multiplayer Teams:** Support for multiple teams with numerous players on each.
* **Unique Identifiers:** Ensuring game, team, and player names are unique for easy tracking and identification.
* **Singleton Game Instance:** Only one game can exist in memory at once, ensuring consistency and avoiding conflicts.
* **Web-Based Accessibility:** Leveraging web technologies to enable cross-platform play.

## [Design Constraints](#_2et92p0)

Developing a web-based game in a distributed environment presents certain design constraints that need to be addressed:

* **Network Latency and Connectivity:**
  + Real-time interactions, such as image rendering and guess submission, must be designed to be resilient to varying network conditions. Strategies like buffering and optimistic updates may be employed to mitigate the impact of latency.
* **Concurrency and Synchronization:**
  + With multiple teams playing simultaneously, mechanisms must be in place to handle concurrent requests and ensure data consistency.
* **Scalability:**
  + The system should be able to handle an increasing number of concurrent games and players without compromising performance. A scalable architecture, with load balancing and potential database sharding, might be necessary to accommodate growth.
* **Browser Compatibility:**
  + The game must function correctly across different web browsers and their varying implementations of web standards. Rigorous testing and adherence to widely supported technologies are crucial.
* **Security:**
  + Measures must be in place to protect against common web vulnerabilities. Input validation, output sanitization, and secure authentication mechanisms are essential.

## [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)

This UML class diagram depicts the initial structure of The Gaming Room's game application domain model. It lays out the key classes and their relationships, providing a foundation for implementing the game's core functionality.

**Class Breakdown:**

* **Entity:** An abstract base class serving as a blueprint for other classes (Game, Team, Player) to inherit common properties (id, name) and behaviors (getters, toString).
* **ProgramDriver:** Contains the main() method, as the application's entry point. It utilizes the GameService.
* **SingletonTester:** A class to test the singleton implementation of GameService.
* **GameService:** The central class managing the game's lifecycle. It holds a list of active Game instances and manages the creation of new games, teams, and players. It ensures only one game instance exists at a time (singleton pattern) and generates unique IDs.
* **Game:** Represents a single game session. It maintains a list of participating Teams.
* **Team:** Represents a team within a game. It holds a list of Players associated with it.
* **Player:** Represents an individual player participating in a team.

**Relationships:**

* **Inheritance:** Game, Team, and Player inherit from Entity, promoting code reusability.
* **Association:**
  + GameService and Game have a one-to-many relationship (GameService manages multiple Games).
  + Game and Team have a one-to-many relationship (Game has multiple Teams).
  + Team and Player have a one-to-many relationship (Team has multiple Players).
* **Uses:** SingletonTester uses GameService to verify its singleton behavior.

**Object-Oriented Principles Demonstrated:**

1. **Encapsulation:** Each class encapsulates its data and behavior, providing controlled access through methods (e.g., getters in Entity).
2. **Inheritance:** Game, Team, and Player inherit common attributes and methods from Entity, reducing redundancy and enhancing code maintainability.
3. **Singleton Pattern:** GameService is designed as a singleton, ensuring only one instance exists, crucial for managing a single game at a time.
4. **Association:** Relationships between classes model the real-world connections between game entities (games have teams, teams have players), facilitating interaction and data management.

**How the Diagram Fulfills Software Requirements:**

* **Multiple teams and players:** The relationships between Game, Team, and Player support having one or more teams with multiple players.
* **Singleton game instance:** The singleton design of GameService guarantees only one game exists in memory.
* **Unique identifiers:** GameService generates unique IDs for games, teams, and players, enabling precise tracking and identification.

**"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 servers are less common for large-scale web applications. While macOS Server exists, it's generally less cost-effective than Linux. | Linux is a highly popular choice for web servers due to its stability, security, and cost-effectiveness (no OS licensing costs). It offers a wide range of robust server software (Apache, Nginx) and a vast ecosystem of open-source tools. | Windows is also a common choice, particularly for companies already invested in the Microsoft ecosystem. Windows Server offers good performance and support but comes with licensing costs. | Mobile devices are not appropriate for server-side applications since servers are always-on systems, often require supporting many users simultaneously and as a result require significantly more resources than a mobile device typically has. |
| **Client Side** | Cross-browser compatibility testing is crucial to ensure consistent functionality and user experience across all mainstream browsers such as Safari, Chrome, Firefox, etc. It is important to test for varying screen sizes and resolutions. | Cross-browser compatibility testing is crucial to ensure consistent functionality and user experience across all mainstream browsers such as Chrome, Firefox, etc. It is important to test for varying screen sizes and resolutions. | Windows is likely the most common platform for the users. Cross-browser compatibility testing is crucial to ensure consistent functionality and user experience across all mainstream browsers such as Edge, IE, Chrome, Firefox, etc. It is important to test for varying screen sizes and resolutions. | Mobile development requires attention to touch-optimized interfaces, smaller screens and efficient performance on devices with varying capabilities, which can increase time and cost of the project as well as more niche technical skills needed to develop for mobile platforms. |
| **Development Tools** | VSCode and Eclipse are good cross-platform choices. Xcode on Macs provides unique tooling specially for mobile app development. Given this project is for a web-based application, the userbase will be using a browser like Safari or Chrome where the application should be largely platform agnostic. Javascript and HTML5 and frameworks such as React or Angular would be most appropriate. | VSCode and Eclipse are good cross-platform choices. Given this project is for a web-based application, the userbase will be using a browser like Chrome where the application should be largely platform agnostic. Javascript and HTML5 and frameworks such as React or Angular would be most appropriate. | VSCode, Visual Studio Eclipse are good cross-platform choices. Given this project is for a web-based application, the userbase will be using a browser like Edge or Chrome where the application should be largely platform agnostic. Javascript and HTML5 and frameworks such as React or Angular would be most appropriate. | Frameworks such as React-Native and Flutter allow developing cross-platform mobile applications where a single version of the code can be used the various mobile device types (android or IOS).  VSCode is an appropriate tool that can be used by engineers using different personal devices. Xcode on mac can be used if they use a Mac device for development. |

## Recommendations

**1. Operating Platform:**

**Recommendation:** Linux (specifically, a distribution like Ubuntu Server)

**Rationale:**

* **Cost-effective:** Linux is open source, eliminating licensing costs which is important for a growing business.
* **Highly stable and reliable:** Known for its robustness and uptime, crucial for a game server.
* **Flexibility and customization:** Allows for fine-grained control over the system, ideal for optimizing performance.
* **Strong community support:** Large and active community provides extensive documentation and troubleshooting resources.
* **Excellent security features:** Offers robust security mechanisms and a strong track record in server environments.

**2. Operating System Architecture:**

Linux follows a **monolithic kernel** architecture. This means that the entire operating system (kernel, device drivers, file system, etc.) runs in a single address space.

* **Benefits:**
  + Efficiency: Faster execution due to reduced overhead from context switching.
  + Simplicity: Easier to develop and maintain.
* **Potential Drawback:** A bug in one part of the kernel can potentially bring down the entire system. However, Linux has a mature and stable kernel, minimizing this risk.

**3. Storage Management:**

**Recommendation:** ZFS (Zettabyte File System)

**Rationale:**

* **Data integrity:** ZFS is known for its advanced data integrity features, including checksum creation and copy-on-write, which protect against data corruption. This is critical for user data and game files.
* **Scalability:** ZFS can handle massive amounts of storage, allowing for easy expansion as "Draw It or Lose It" grows.
* **Snapshots and clones:** Provides efficient snapshots and cloning capabilities for backups and testing.

**4. Memory Management:**

Linux utilizes several memory management techniques:

* **Paging:** Divides physical memory into fixed-size blocks (pages) and virtual memory into pages of the same size. This allows a process to be stored in memory non-contiguously.
* **Swapping:** Moves inactive pages from RAM to a designated swap space on the hard disk, freeing up RAM for active processes.
* **Demand paging:** Loads pages into RAM only when they are needed.
* **Copy-on-write (COW):** Allows multiple processes to share the same pages of memory until one of them modifies a page.

**How these apply to "Draw It or Lose It":**

* Efficient memory allocation for game assets, user data, and game logic.
* Prevents the game from crashing if it requires more memory than physically available.
* Optimizes memory usage for better performance and responsiveness.

**5. Distributed Systems and Networks:**

To enable communication between different platforms (e.g., Android, iOS, web browsers), "Draw It or Lose It" can be designed as a **client-server** distributed system.

* **Components:**
  + **Game Server (Linux):** Central server running the game logic, managing user accounts, and handling game sessions.
  + **Game Clients (Various platforms):** Applications on user devices (phones, tablets, computers) that interact with the server.
* **Network:** The clients and server communicate over the internet using a reliable protocol like TCP/IP.
* **Dependencies and Considerations:**
  + **Connectivity:** Reliable internet connection is crucial for both the server and clients.
  + **Outages:** Implement redundancy (e.g., multiple servers, backup power) to mitigate the impact of server outages.
  + **Latency:** Minimize latency (delay) by optimizing network communication and server performance.

**6. Security:**

* **Data encryption:** Encrypt sensitive data (usernames, passwords, game data) both in transit (using TLS/SSL) and at rest (using disk encryption).
* **Access control:** Implement strong authentication and authorization mechanisms to restrict access to user data and prevent unauthorized actions.
* **Regular security updates:** Keep the Linux server and game clients updated with the latest security patches.
* **Firewalls:** Use firewalls to block unauthorized network traffic to the server.
* **Data backups:** Regularly back up user data and game data to protect against data loss.
* **Secure coding practices:** Develop the game server and clients using secure coding practices to prevent vulnerabilities.
* **SELinux (Security-Enhanced Linux):** Provides mandatory access control to confine processes and limit their potential damage.
* **Regular security audits:** Conduct regular security audits and penetration testing to identify and address potential vulnerabilities.

By following these recommendations, The Gaming Room can create a secure, scalable, and high-performing gaming experience for "Draw It or Lose It" across multiple platforms.