

# Draw It or Lose It Web Application

# **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 | 11/29/2024 | Caeli firth | Added evaluation and recommendations for Project Two, including detailed analysis of operating platforms, distributed systems, and security measures. |

**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 seeks to expand their Android-based game, **Draw It or Lose It**, into a web-based application to support multiple platforms. This transition requires a scalable, secure, and efficient solution to meet their business needs, including managing game sessions, maintaining unique game and team names, and ensuring compatibility across various operating systems. To achieve this, the software design will leverage **object-oriented programming (OOP)** principles and implement design patterns such as **singleton** and **iterator**, ensuring reliability and maintainability. Our approach addresses the unique challenges of a distributed, web-based environment while enabling seamless user experiences across devices.

## Requirements

*The Gaming Room's primary requirements are:*

* ***Business Requirements****:*
  + *Support multiple teams and players in a game.*
  + *Ensure all game and team names are unique.*
  + *Provide a web-based experience accessible from various platforms.*
* ***Technical Requirements****:*
  + *Ensure only one game instance exists in memory at a time.*
  + *Implement design patterns to manage data efficiently and securely.*
  + *Build a responsive and platform-independent web application.*

## [Design Constraints](#_2et92p0)

Key constraints for this project include:

1. **Distributed Web Environment**:
   * The application must handle multiple concurrent users across platforms, requiring robust session and state management.
   * Implication: Scalable server architecture and efficient database access mechanisms are critical.
   * Implement asynchronous processing to handle high user traffic, using frameworks like Spring Boot with reactive programming.
2. **Unique Identifiers**:
   * Game, team, and player names must be unique, ensuring no conflicts during gameplay.
   * Implication: Use validation and efficient data search techniques such as the iterator pattern.
   * A hash table or UUIDs will be used to enforce uniqueness for names, improving retrieval speed and avoiding collisions.
3. **Singleton Game Service**:
   * Only one instance of the GameService class can exist at a time.
   * Implication: Ensures memory efficiency and prevents data duplication but requires careful thread safety in concurrent environments.
   * Thread-safe singleton implementation will prevent race conditions in multi-threaded environments, ensuring consistent service.
4. **Cross-Platform Compatibility**:
   * The game must run on Linux, Mac, Windows, and mobile devices.
   * Implication: Frameworks and tools must support multi-platform development without compromising performance.
   * The application will use responsive web design principles and tools like Bootstrap to ensure adaptability to different screen sizes and platforms.

## [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 provided UML class diagram outlines the following:

* **Entity Class**:
  + A base class that holds common attributes (id, name) shared by all entities.
* **Game Class**:
  + Inherits from Entity. Contains a list of teams and manages game-specific logic.
* **Team Class**:
  + Inherits from Entity. Holds a list of players and tracks team-specific details.
* **Player Class**:
  + Inherits from Entity. Represents individual participants in the game.

**Object-Oriented Principles Demonstrated**:

1. **Inheritance**:
   * Reduces code duplication by defining shared attributes in Entity and extending them to child classes.
2. **Encapsulation**:
   * Protects data integrity by providing controlled access to attributes through getters and setters.
3. **Polymorphism**:
   * Simplifies interactions by allowing the same operations (e.g., getName()) to work across all derived classes.

These principles enhance code maintainability and scalability, ensuring the design meets client requirements efficiently.

**"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 but can handle web-based applications. The Unix-based OS offers good stability, though it may require more configuration and maintenance than Linux. | Linux is widely used in web servers due to its stability, open-source nature, and strong performance. Low licensing cost makes it ideal for scaling. | Windows Server works well for applications needing .NET, though licensing can be expensive. Compatibility with Windows services may offer ease in corporate settings. | Rarely used for server hosting, but mobile devices could support light local hosting in unique cases. |
| **Client Side** | Supporting Mac as a client platform ensures compatibility with Safari, Chrome, and Firefox. Development time might increase if needing to test specifically on macOS. | Linux desktops are less common among end-users, but testing should consider Chrome and Firefox compatibility. | Windows has a broad user base, so supporting it is essential. Popular browsers like Chrome, Firefox, and Edge should be tested. | Ensuring compatibility with mobile browsers and screen responsiveness is key. The game needs to work well on Android and iOS browsers. |
| **Development Tools** | Languages and tools like Java, IntelliJ IDEA, and Xcode for testing could be used. Web-based frameworks are typically cross-compatible. | Eclipse, NetBeans, and command-line tools are Linux-friendly. Frameworks like Node.js, Java, and Spring Boot can run on Linux servers efficiently. | Visual Studio and .NET frameworks are strong for Windows development. Java is also supported for cross-platform compatibility. | For mobile optimization, tools like React Native, Swift, and Kotlin allow for development compatible with the game’s web-based format. |

## 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:**
   1. Recommend Linux as the server platform due to its stability, cost-efficiency, and strong support for web-based frameworks.
2. **Operating Systems Architectures:**
   1. Linux uses a modular architecture ideal for high-performance web applications. It supports multi-threading and efficient memory management critical for scalability.
3. **Storage Management:**
   1. Use a relational database like PostgreSQL to handle structured data for games, teams, and players. It ensures reliability and supports distributed systems.
4. **Memory Management:**
   1. Linux employs robust virtual memory techniques, allowing efficient allocation for game instances and ensuring smooth performance under heavy loads.
5. **Distributed Systems and Networks:**
   1. Implement RESTful APIs to enable communication between components. Use load balancers to handle traffic spikes and ensure redundancy to mitigate outages.
6. **Security:**
   1. Employ HTTPS for secure data transmission.
   2. Use OAuth 2.0 for user authentication.
   3. Implement database encryption and firewall rules to protect sensitive user information.