

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 12/15/24 | Yerlan Baiyekeshov | Updated recommendations. |

**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 is planning to expand its game "Draw It or Lose It" by developing a web-based application. This expansion aims to increase accessibility while maintaining the engaging and collaborative nature of the original game.

The client has outlined the following key software requirements for the game application:

1. A game must support one or more teams participating.
2. Each team will consist of multiple players.
3. Game and team names must be unique, allowing users to verify name availability when creating or joining a team.
4. Only one instance of the game should exist in memory at any given time, achieved through unique identifiers for each instance of a game, team, or player.

The proposed solution leverages object-oriented programming principles, such as inheritance, encapsulation, and the Singleton design pattern, to design a scalable and efficient architecture. It will also employ modern development frameworks to ensure security, platform compatibility, and real-time responsiveness.

By addressing these requirements, this project will deliver a robust application that meets the client’s needs.

## Requirements

To ensure the successful development of the web-based version of "Draw It or Lose It," the following business and technical requirements must be addressed:

#### Business:

1. **Cross-Browser Compatibility**: Provide a seamless user experience across multiple platforms (Linux, Mac, Windows, and IOS devices).
2. **Performance**: Support real-time gameplay for multiple users simultaneously.
3. **Security:** Ensure secure handling of user information and authentication.
4. **Scalability**: Maintain a scalable application structure to accommodate future growth in the user base.

#### Technical:

1. Utilize object-oriented programming principles to create reusable, maintainable code.
2. Implement secure communication protocols to protect user data.
3. Ensure compatibility with different operating systems and browsers.
4. Optimize the application for performance, minimizing latency during real-time interactions.
5. Incorporate a componential design to allow for easier updates and feature expansions.

## [Design Constraints](#_2et92p0)

Developing a web-based version of "Draw It or Lose It" introduces several design constraints that need to be addressed to ensure compatibility, security, and performance:

* 1. **Cross-Browser Compatibility**:

The application must function seamlessly across various web browsers, including Chrome, Firefox, Safari, and Microsoft Edge.

**Constraint**: Differences in browser behavior and rendering engines may necessitate thorough testing and debugging to ensure a uniform user experience.

* 1. **Security**:

The application must protect user data through encryption and secure authentication mechanisms.

**Constraint**: Implementing robust security measures may increase development time and require regular updates to address new vulnerabilities.

* 1. **Scalability**:

The system must handle growing user numbers as the game gains popularity, requiring scalable infrastructure.

**Constraint:** Designing for scalability may involve additional costs for hosting and infrastructure.

* 1. **Performance**:

The game must minimize latency and ensure real-time responsiveness during gameplay.

**Constraint: P**erformance optimization may require advanced techniques, such as caching and load balancing.

## [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 diagram outlines the relationships between the key classes in the system and demonstrates the use of object-oriented programming principles.

1. **Entity**:

* Acts as a base class for Game, Team, and Player.
* Contains common attributes like id and name, ensuring consistency and reducing code duplication.
* Demonstrates inheritance, as subclasses (Game, Team, and Player) inherit id and name properties.

2. **GameService**:

* A singleton class responsible for managing the overall state of the application.
* Contains a list of Game objects (games) and tracks identifiers for games, players, and teams.
* Methods include addGame, getGame, and getGameCount, which enable efficient game management.

3. **Game**:

* Represents an individual game instance.
* Contains a list of Team objects (teams), demonstrating a one-to-many relationship between Game and Team.
* Includes methods for adding teams and retrieving game details.

4. **Team**:

* Represents a team within a game.
* Contains a list of Player objects (players), demonstrating a one-to-many relationship between Team and Player.
* Includes method for adding players addPlayers.

5. **Player**:

* Represents an individual player.
* Inherits common attributes from the Entity class, such as id and name.

6. **ProgramDriver** and **SingletonTester**:

* The ProgramDriver class initializes and tests the application.
* The SingletonTester class verifies the singleton design pattern used in GameService.

**"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** | Stable environment, good for development but higher costs for hosting. | Cost-effective and reliable hosting option; highly scalable. | Compatible with many enterprise tools but higher licensing costs. | Limited server capabilities. Cloud services like AWS or Firebase are often used for hosting. |
| **Client Side** | High-quality graphics rendering; user-friendly interface. | Minimal UI compatibility issues; needs thorough cross-browser testing. | Broad software support, but performance may lag slightly in multi-platform environments. | Crucial to ensure lightweight performance on devices with limited hardware. |
| **Development Tools** | Ruby, Python, Swift. Xcode for mobile apps. | Apache, NGINX, Docker. Proffered for server development. | PHP,  Visual Studio, .NET framework. | Android Studio for Android development; Xcode for iOS; React Native for cross-platform mobile apps |

## 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**:

Linux is recommended for hosting due to its cost-effectiveness, scalability, and robust support for server-side technologies. Leveraging a Linux-based environment enables efficient resource utilization, especially for gaming applications with real-time demands.

1. **Operating Systems Architectures**:

Adopting a **microservices architecture** is essential for scaling the application effectively. Each component (e.g., game logic, user management, and team tracking) can be independently developed, deployed, and scaled. Tools like **Docker** and **Kubernetes** should be used to containerize services, ensuring consistent deployment across environments and simplifying maintenance.

1. **Storage Management**:

The storage strategy should focus on scalability and reliability:

* Use **AWS S3** for object storage, which provides high durability, automatic scaling, and seamless integration with other AWS services like EC2.
* For real-time data (e.g., active games and user sessions), implement **Redis** or **Memcached** to cache frequently accessed data, minimizing latency.

1. **Memory Management**:

Efficient memory management is critical to support real-time gameplay:

* Use garbage collection techniques to optimize memory allocation and prevent leaks, particularly in the backend server.
* Integrate **in-memory databases** (e.g., Redis) for fast read/write operations, which are crucial for gaming scenarios with high concurrency.

1. **Distributed Systems and Networks**:

The game's distributed architecture should prioritize fault tolerance, scalability, and low latency:

* Use **RESTful APIs** to facilitate communication between system components, ensuring flexibility and compatibility with various clients.
* Employ **load balancers** to distribute user requests evenly across servers, preventing bottlenecks during peak usage.
* Implement **Content Delivery Networks** to deliver static assets (e.g., images and UI components) to users quickly and reliably, regardless of location.

1. **Security**:

Security measures should protect user data and application integrity:

* Enforce **HTTPS/SSL** encryption for all data in transit.
* Hash user passwords using strong algorithms (e.g., bcrypt) and implement **two-factor authentication (2FA)** for account protection.
* Regularly audit the application for vulnerabilities and update libraries and dependencies to mitigate security risks.