

# "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 |
| --- | --- | --- | --- |
| 3.0 | 12/15/2024 | Michael Geske | |  | | --- | | Initial draft of the software design document, including executive summary, evaluation of development platforms, domain model explanation, and recommendations for system architecture and security. | |

## [Executive Summary](#_sbfa50wo7nsh)

"Draw It or Lose It" is a web-based gaming application inspired by the classic game show "Win, Lose or Draw," where teams compete to guess puzzles based on progressively revealed drawings. This project aims to create an engaging game platform that supports multiple teams and players, ensuring smooth gameplay and meeting all client requirements. The solution will implement a centralized game instance with unique identifiers for all entities (game, team, and players) to maintain integrity and prevent conflicts. This document outlines the design constraints, domain model, and technical considerations required to achieve these goals.

## [Design Constraints](#_2et92p0)

The application will operate in a web-based distributed environment, which presents the following design constraints:

1. **Concurrency**: Only one active game instance can exist at a time. This requires careful synchronization and management of game states.
2. **Uniqueness**: Game and team names must be unique. Implementing a robust name-checking mechanism is essential.
3. **Scalability**: The system must handle multiple teams, each with several players, ensuring consistent performance regardless of load.
4. **Distributed Environment**: Hosting and communication between clients and the server must remain efficient, leveraging RESTful APIs or WebSockets for real-time updates.
5. **Cross-Platform Support**: Compatibility with multiple platforms (desktop and mobile) demands adherence to responsive design principles.
6. **Security**: Safeguarding user data and ensuring secure communication across platforms.

Implications: These constraints necessitate a well-structured architecture, adherence to object-oriented principles, and the use of reliable frameworks and libraries for web development.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram provided outlines the following core components:

1. **Game**: Represents the entire game instance, with unique identifiers to prevent duplicates.
2. **Team**: Each team belongs to a game and comprises multiple players. The team name must be unique within the game.
3. **Player**: Represents individual users, with unique identifiers to distinguish between players.

**Object-Oriented Principles**:

* **Encapsulation**: Each class encapsulates its attributes (e.g., game state, player list) and behaviors (e.g., name validation).
* **Inheritance**: The Entity base class ensures common attributes (e.g., unique ID) and behaviors are inherited by all subclasses.
* **Polymorphism**: Potentially used for extending the functionality of classes (e.g., different game types).
* **Abstraction**: Simplifies complex operations like game state management, exposing only necessary interfaces to other components.

**"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** | macOS provides excellent stability and compatibility for web-based applications. However, its closed ecosystem limits flexibility and can increase deployment costs due to expensive hardware requirements. | Linux is ideal for hosting web-based applications due to its open-source nature, flexibility, and extensive community support. It offers superior performance for distributed systems at minimal cost. | Windows servers offer user-friendly tools, broad software support, and easy integration with Microsoft services, but they may require expensive licenses and have higher resource overhead. | Hosting server-side services on mobile devices is uncommon due to limited resources and battery constraints. However, they can function as edge devices in distributed architectures. |
| **Client Side** | macOS client applications require careful testing due to compatibility with Apple hardware. Costs can rise for small teams without access to Apple devices for testing. | Linux as a client platform is limited by its lower adoption among end-users. Applications for Linux often target power users or niche markets | Windows clients are widely adopted, making this a priority for development. It’s easy to deploy applications on this platform, but compatibility testing is required for legacy systems. | Mobile devices demand responsive design and cross-platform frameworks. They must balance performance with accessibility across Android and iOS. |
| **Development Tools** | Development on macOS often involves Swift and Objective-C, with tools like Xcode for client-side applications. For server-side, Node.js or Django are commonly used. | Linux supports a wide range of programming languages and tools like Python, Java, and C++. IDEs like Eclipse and VS Code are widely used. | Windows offers .NET for development, Visual Studio as the primary IDE, and compatibility with multiple programming languages like C#, Python, and JavaScript. | Mobile development requires tools like Android Studio (for Android), Xcode (for iOS), or cross-platform frameworks like Flutter or React Native. |

## Recommendations

1. **Operating Platform**  
   **Recommended Platform**: Linux for server-side hosting and cross-platform frameworks (e.g., React Native) for client-side development.  
   **Rationale**: Linux is highly stable, scalable, and cost-effective for hosting, while cross-platform tools enable consistent performance on diverse devices.
2. **Operating Systems Architectures**  
   Linux supports multi-threaded, distributed applications, making it suitable for web-based environments. Its modularity enables efficient resource allocation and real-time processing, crucial for the game’s real-time rendering and communication.
3. **Storage Management**  
   Use cloud-based solutions scalability, reliability, and cost-efficiency.
4. **Memory Management**  
   Memory-intensive operations, such as rendering drawings, can use Linux’s efficient virtual memory management. Technologies to optimize memory allocation for containerized services will be integral.
5. **Distributed Systems and Networks**  
   Utilize applications for real-time updates between the clients and the servers. Implement load balancing to ensure uptime and redundancy to handle connectivity outages.
6. **Security**

* Encrypt data using TLS for secure communication.
* Implement authentication mechanisms for access control.
* Use secure storage protocols to safeguard user data.
* Regularly patch and update the system to protect against vulnerabilities.