

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 | <08/24/2024> | <Khori Cohen-Welch> | <Updated – 3.0 - development requirements for all Operating Systems> |

**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 needs a web-based game called **"Draw It or Lose It,"** where teams compete to guess images from a library of stock drawings. The game must support multiple teams, ensure unique names for games, teams, and players, and only allow one game instance in memory at a time. To meet these needs, we'll develop a Java application using the Singleton pattern to manage the game instance and the iterator pattern to handle game, team, and player collections efficiently. We'll have to create a base Entity class to standardize attributes across the game's components, making the system easy to maintain and expand. This design will provide a solid foundation for building a game that is both user-friendly and easy to manage.

## Requirements

The Gaming Room's game application must meet the following requirements:

1. **Team Participation**: The game should support one or more teams.
2. **Player Assignment**: Each team must have multiple players.
3. **Unique Names**: Game and team names must be unique to avoid duplication.
4. **Single Instance**: Only one instance of the game should exist in memory at any time.
5. **Identifier Generation**: Unique identifiers must be assigned to each game, team, and player.

These requirements are essential for ensuring smooth gameplay and efficient management of game data.

## [Design Constraints](#_2et92p0)

1. Web Capable - The game needs to work smoothly on various web browsers and devices. This means choosing the right technologies and ensuring consistent performance across platforms.
2. Multiplayer - Multiple users might interact with the game simultaneously, so the design must handle simultaneous actions without conflicts.
3. Performance - The application must be fast and responsive, minimizing load times and ensuring a smooth user experience.

## [System Architecture View](#_ilbxbyevv6b6)

For "Draw It or Lose It," the system architecture is designed to ensure seamless performance and scalability across various platforms. Here's a summary of how it’s structured:

The application is divided into several key components:

1. **Client Tier:** Players interact with the game through web browsers or mobile apps. This tier handles the user interface and communicates with the server to send game requests and receive updates.
2. **Application Server Tier:** This is where the core game logic resides. It processes player actions, manages game sessions, and updates game states. Hosted on a Linux server, it ensures reliability and can handle high traffic.
3. **Database Tier:** All game data, including player profiles, statistics, and image metadata, is stored here. Depending on the requirements, either a relational database (like MySQL) or a NoSQL database (like MongoDB) is used for flexibility and scalability.
4. **Storage Tier:** Given the game's need for high-definition images, a cloud-based storage service such as Amazon S3 is used. This allows for scalable, reliable storage that handles the large volume of image files efficiently.
5. **Network Tier:** Load balancers are employed to distribute traffic across multiple servers, ensuring that the application remains responsive even during peak times. Redundant network connections and failover mechanisms are in place to maintain service continuity.

In summary, the architecture allows "Draw It or Lose It" to run smoothly by efficiently handling client-server communication, managing data storage and retrieval, and ensuring the system can scale and remain reliable. This setup ensures that players have a seamless experience while the system effectively manages high volumes of data and traffic.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram includes these main components:

1. **Entity**: The base class with id and name attributes, shared by all entities.
2. **Game**: Represents a game and holds a list of teams.
3. **Team**: Represents a team and holds a list of players.
4. **Player**: Individual player, inheriting from Entity.
5. **GameService**: Manages game instances and ensures only one instance exists (Singleton pattern).

**Key Principles**:

* **Inheritance**: Game, Team, and Player all inherit from Entity, sharing common attributes.
* **Composition**: Game and Team manage multiple Team and Player instances, respectively.
* **Singleton**: GameService ensures there's only one instance to manage the game.

These principles help keep the game structured, ensure unique identifiers, and manage data 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 can host a web-based application, but it is less commonly used for server environments compared to Linux and Windows. It supports server-based deployment but comes with higher hardware costs and licensing fees for macOS Server. Mac is generally not the preferred choice for high-scale web hosting. | For Linux, it is highly suitable for hosting a web-based application due to its robust performance and scalability. Linux supports server-based deployment efficiently and is ideal for managing thousands of players. It is cost-effective, with no licensing fees for the OS itself, although there might be costs for support or additional software. | Windows is well-suited for hosting web applications, offering extensive support and user-friendly features. It supports server-based deployment and is commonly used in enterprise settings. However, it requires licensing fees for Windows Server editions, which can add to the overall cost of deployment. | Mobile Devices are not suitable for hosting web applications due to their limited processing power and storage. They are intended for accessing applications rather than hosting server-side software. |
| **Client Side** | To ensure compatibility with Mac, the application must be developed using modern web technologies such as HTML5, CSS3, and JavaScript. This requires thorough testing across different versions of macOS and major web browsers to ensure a consistent user experience. The development process involves cost for tools and testing, time for cross-browser adjustments, and expertise in web development to handle any macOS-specific quirks. | For Linux, the application should also utilize HTML5, CSS3, and JavaScript to ensure it runs smoothly across various Linux distributions and web browsers. Development will require testing on different Linux environments to address any compatibility issues. The considerations include budget for development tools and testing, time for ensuring compatibility with diverse Linux setups, and expertise in web technologies and Linux-specific issues. | On Windows, the application must be compatible with different versions and web browsers. Using HTML5, CSS3, and JavaScript is crucial for ensuring a responsive and functional interface. Development involves cost for tools and potential licensing, time for testing across Windows versions, and expertise in handling Windows-specific browser behavior. | For mobile platforms (iOS and Android), the application can be developed using either native approaches or cross-platform frameworks like React Native or Flutter. This requires adaptation for touch interfaces and performance optimization. The development process will involve higher costs due to the need for mobile development tools and testing on various devices. Expertise in mobile app development is essential, along with additional time for ensuring the application performs well across different screen sizes and operating systems. |
| **Development Tools** | Development on Mac primarily uses Swift and Objective-C with Xcode as the main IDE. Xcode requires a paid macOS Developer Program membership. This setup needs a team familiar with macOS development and may involve additional costs and resources for Mac-specific issues. | On Linux, languages like C, Python, and Java are used, with IDEs such as Eclipse and Visual Studio Code. These tools are mostly open-source with no additional licensing fees. Development requires expertise in Linux environments, which might mean extra training or a specialized team. | Windows development involves languages like C#, C++, and Java, with Visual Studio as the main IDE. Visual Studio has licensing costs, especially for professional editions. Developers need to be experienced with Windows tools, and you might need multiple team members to handle various Windows-specific tasks. | For mobile platforms, Swift is used for iOS and Kotlin or Java for Android, with Xcode and Android Studio as the primary IDEs. Cross-platform tools like React Native or Flutter are also options. Xcode and Android Studio are free, but cross-platform tools might have costs. This requires expertise in mobile development and could mean separate teams for iOS and Android, or a combined team if using cross-platform tools. |

## 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**: For expanding "Draw It or Lose It" to other computing environments, a Linux-based server platform is highly recommended. Linux offers robust performance, stability, and scalability, which are essential for hosting a web-based application capable of handling thousands of concurrent players. Its open-source nature also means cost savings on licensing fees, making it a budget-friendly option for The Gaming Room.
2. **Operating Systems Architectures**: Linux operates on a modular architecture that includes a monolithic kernel, which manages system resources and hardware. It supports multi-user environments and is known for its efficient process management and strong security features. The architecture's ability to handle multiple processes and users simultaneously makes it ideal for running a high-traffic web application.
3. **Storage Management**: For managing the 1.6 GB of image files and additional game data, a distributed storage system like Amazon S3 (Simple Storage Service) is recommended. It provides scalable storage solutions with high availability and durability, ensuring that game data is securely stored and easily accessible. S3’s integration with AWS services also supports efficient data management and backup.
4. **Memory Management**: Linux employs several memory management techniques that are beneficial for "Draw It or Lose It." These include paging, segmentation, and efficient memory allocation strategies. The use of virtual memory allows the system to handle large data loads without impacting performance. Additionally, Linux’s built-in tools for memory profiling and garbage collection help optimize application performance by managing memory resources effectively.
5. **Distributed Systems and Networks**: To enable communication between various platforms, "Draw It or Lose It" should utilize a RESTful API for client-server interactions. This allows seamless integration across different devices and operating systems. For network connectivity, implementing load balancers and ensuring fault tolerance with redundant servers will help manage high traffic and maintain availability. It’s also important to consider network security measures to protect against potential outages and security breaches.
6. **Security**: To safeguard user information across various platforms, it’s crucial to implement encryption both in transit and at rest. Using HTTPS for secure communication between clients and servers ensures that data is protected from eavesdropping and tampering. Additionally, employing strong authentication mechanisms and regular security updates will help prevent unauthorized access and protect user data from potential threats.

By addressing these aspects, The Gaming Room can ensure that "Draw It or Lose It" performs efficiently, scales effectively, and maintains high security across all platforms.