

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 | 05/26/24 | Andrei Shostak | Initial creation of the software design document |

**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 aims to expand its game, Draw It or Lose It, from a solely Android-based application to a web-based application that serves multiple platforms. The game involves teams competing to guess a drawing rendered from a stock image library. To achieve this, the application must support multiple teams and players, ensure unique game and team names, and maintain only one instance of the game in memory at any given time. The proposed solution involves developing a software design that leverages object-oriented programming principles and design patterns, such as the singleton and iterator patterns, to meet these requirements effectively.

## Requirements

Business Requirements:

* Expand Draw It or Lose It to a web-based application.
* Support multiple platforms beyond Android.
* Enable multiplayer functionality with multiple teams and players.
* Ensure game and team names are unique.

Technical Requirements:

* Implement a singleton pattern to ensure only one instance of the game exists in memory.
* Use the iterator pattern for efficient game and team management.
* Create a base class to hold common attributes and behaviors.
* Ensure all teams and players have unique identifiers.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment presents several design constraints:

* Scalability: The application must handle multiple users simultaneously, requiring efficient resource management and load balancing.
* Performance: The game must render images and handle user inputs in real-time, demanding optimized code and low-latency network communication.
* Security: User data must be protected, necessitating secure authentication, data encryption, and secure communication protocols.
* Compatibility: The application must be compatible with various operating systems and devices, influencing the choice of development tools and technologies.

These constraints necessitate careful planning and implementation of the software architecture to ensure a robust, efficient, and secure application.

## [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 class diagram represents the structure of the game application, showcasing the relationships between various classes and their responsibilities:

* Entity Class: A base class holding common attributes (id and name) and methods (getId(), getName(), toString()). This class promotes code reuse and simplifies maintenance.
* GameService Class: Manages the overall game logic, including game creation and retrieval. Implements the singleton pattern to ensure a single instance.
* Game Class: Represents a game with a list of teams. Inherits from Entity and adds specific methods to manage teams.
* Team Class: Represents a team with a list of players. Inherits from Entity and includes methods to manage players.
* Player Class: Represents a player. Inherits from Entity.

Object-oriented principles demonstrated in the diagram:

* Inheritance: The Game, Team, and Player classes inherit from the Entity class, promoting reuse and reducing redundancy.
* Encapsulation: Each class manages its own data and methods, providing a clear interface for interaction.
* Polymorphism: The Entity class methods can be overridden by subclasses, allowing for flexible and dynamic behavior.

**"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 known for their stability and security but can be expensive. They provide a good environment for web-based applications but are less commonly used in enterprise environments. | Linux is highly preferred for server environments due to its robustness, security, flexibility, and cost-effectiveness. It supports various web servers like Apache and Nginx, making it an ideal choice for hosting web applications. | Windows servers offer ease of use and integration with other Microsoft services. They are widely used but can be more vulnerable to security threats and require regular updates. | Hosting a web application directly on mobile devices is not typical. Mobile devices are generally used as clients accessing the application hosted on a server. |
| **Client Side** | Mac provides a consistent development environment but has a smaller user base compared to Windows. Developing for Mac ensures high performance and security. | Linux offers a powerful development environment with a variety of tools. However, the user base is smaller, and there may be compatibility issues with certain applications. | Windows has the largest user base and a wide range of development tools, making it essential to support. However, it can introduce more security and maintenance challenges. | Mobile devices are essential to support due to the large user base. Development must consider different operating systems (iOS, Android) and various screen sizes and resolutions. |
| **Development Tools** | Development on Mac can use Xcode, Swift, and other tools like IntelliJ IDEA and Eclipse for cross-platform development. | Linux supports a wide range of programming languages and tools such as GCC, Eclipse, and VS Code. It is highly flexible and customizable for development. | Windows offers tools like Visual Studio, .NET, and a wide range of third-party IDEs. It supports a broad spectrum of programming languages and frameworks. | Mobile development requires tools like Android Studio for Android, Xcode for iOS, and cross-platform tools like Flutter and React Native. |

## 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 the web-based application due to its stability, security, and cost-effectiveness.
2. **Operating Systems Architectures**: Linux provides a robust architecture with support for multi-threading, high performance, and efficient resource management.
3. **Storage Management**: Utilize a distributed database system like PostgreSQL or MongoDB to ensure data consistency, scalability, and reliability.
4. **Memory Management**: Linux uses advanced memory management techniques such as virtual memory, swapping, and memory allocation to optimize performance and resource utilization.
5. **Distributed Systems and Networks**: Use RESTful APIs for communication between platforms, ensuring interoperability and scalability. Implement load balancing and failover mechanisms to handle connectivity and outage issues.
6. **Security**: Implement HTTPS for secure communication, use OAuth for secure authentication, and apply data encryption to protect user information both at rest and in transit. Regular security audits and updates are essential to maintaining security across platforms.