

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/22/24 | Patrick Brizuela | Modifications were made to the cover page, document revision history, executive summary, design constraints, and domain model. |
| 1.1 | 10/6/2024 | Patrick Brizuela | Modifications were made to the cover page, document revision history, and evaluations. |
| 1.2 | 10/20/2024 | Patrick Brizuela | Modifications were made to the cover page, document revision history, and 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 purpose of this software design document is to present the proposed solution for developing the web-based game application, "Draw It or Lose It," for our client, The Gaming Room. Inspired by the classic TV game show *Win, Lose, or Draw*, the game allows teams to compete in guessing drawings based on stock images used as clues, with multiple teams participating in over four rounds. Expanding the game to a web-based platform will make it accessible across various devices, offering an enhanced gaming experience. The design aligns with the client's requirements and aims to deliver a user-friendly, engaging application.

## 

## Requirements

1. The game must be accessible via a web-based platform, ensuring compatibility with various devices and operating systems.
2. Each game should allow one or more teams to participate, with the ability to assign multiple players to each team.
3. Game and team names must be unique to avoid conflicts, and users should be able to check name availability during team creation.
4. The application should ensure that only one instance of the game is running in memory at any time, using unique identifiers for games, teams, and players.
5. Each round of the game should have a specific time limit, such as one minute, with drawings gradually revealed, becoming fully visible at the 30-second mark.
6. If a team fails to guess the puzzle within the time limit, other teams should be given one chance each to guess the answer within a 15-second window.

## [Design Constraints](#_2et92p0)

1. **Web-based Distributed Environment:** The game application must be developed for a web-based platform, which presents constraints such as network communication, security, and compatibility across various web browsers and devices.
2. **Unique Names:** The system should enforce unique game, team, and player names to avoid conflicts, ensuring a smooth user experience during game creation and joining.
3. **Single Instance Limitation:** The design must ensure that only one instance of the game service is active in memory at any given time, ensuring proper functionality and preventing system conflicts.

## [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 diagram below depicts the class structure of the *Draw It or Lose It* game application system. **ProgramDriver:** This class serves as the entry point for the program, containing the main() method.  
**SingletonTester:** This class is used to test the Singleton implementation in GameService.  
**Entity:** This parent class represents a generic constructor from which its child classes—Game, Team, and Player—inherit.  
**GameService:** This class manages various game operations, overseeing games, teams, and players while adhering to the Singleton pattern to ensure only one instance of GameService exists. It establishes relationships in a chain of zero to many from Game to Team to Player.  
**Game:** This class manages a game and maintains a list of teams, allowing for team manipulation. It has a zero to many relationships with Team.  
**Team:** Similar to Game, this class focuses on storing and manipulating players, maintaining a zero to many relationships with Player.  
**Player:** This class holds the player ID for use by the parent classes.

In terms of the OOP principles illustrated by this UML diagram:

* **Inheritance** is a core concept, as Entity acts as a base class for multiple subclasses.
* **Abstraction** is demonstrated by Entity, which abstracts common attributes away from the subclasses.
* **Encapsulation** is evident in GameService, where private methods are contrasted with public methods, allowing GameService to manage itself while controlling access to the state of any class through getter and setter methods.
* **Polymorphism** is shown as each child class of Entity overrides the toString method for its specific needs.

**"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** | * Unix-based operating system * Provides a stable and secure platform for hosting web applications * Offers a robust and developer-friendly ecosystem * Higher hardware costs compared to alternatives * Scalability is limited compared to Linux and Windows | | * Open-source operating system offering customization and flexibility * Supports a wide range of software and tools * Highly scalable, with strong stability and security * Some limitations with graphical user interface (GUI) * Potential hardware compatibility issues | * Wide software compatibility and strong developer ecosystem * Broad hardware support * Robust support resources and documentation * More frequently known security vulnerabilities | * Portable design * Touchscreen and gesture-based interface * Limited screen size * Varying hardware capabilities |
| **Client Side** | * Intuitive, user-friendly interface that minimizes the learning curve * Developing and maintaining multiple clients increases development costs and time * Requires diverse expertise for managing multiple platforms | | * Free to use and distribute * Additional costs may arise for hardware and tools * Steeper learning curve * Requires diverse expertise to manage different clients | * Higher licensing costs compared to open-source alternatives * May increase overall project expenses * Can limit flexibility and customization options | * Important considerations include responsive design and connectivity limitations * Utilizes native features such as camera, GPS, and push notifications |
| **Development Tools** | * Commonly used languages include Node.js and JavaScript * Popular IDEs include VSCode and XCode | | * Rich ecosystem with IDEs such as VSCode, Atom, and Sublime Text * Robust command-line interface * Package management systems like apt and yum | * Popular languages for building Windows-based web applications include C# and the .NET framework * Commonly used IDEs are Visual Studio and JetBrains | * Programming languages include Kotlin, Swift, Objective-C, Java, and JavaScript * Development tools include Android Studio and Xcode * Device emulators and simulators are also utilized |

## Recommendations

1. **Operating Platform**: To successfully expand *Draw It or Lose It* to various computing environments, it is strongly recommended to adopt a web-based platform. This approach will make the game accessible on a wide array of devices with web browsers, including desktops, laptops, tablets, and smartphones. By leveraging web technologies, the game can reach a broader audience while ensuring a consistent and seamless experience across different platforms. This strategic move is essential for maximizing user engagement and promoting growth.
2. **Operating Systems Architectures**: It is highly recommended that the architecture for the chosen web-based platform be built around a client-server model using web technologies. On the client side, HTML, CSS, and JavaScript should be utilized to deliver a responsive game interface and handle user interactions efficiently. On the server side, a multi-tier architecture must be implemented, consisting of the presentation, application, and data tiers. This approach will ensure scalability, modularity, and maintainability, providing a solid foundation for long-term success and adaptability across various devices.
3. **Storage Management**: It is strongly recommended to implement a storage management solution that combines a relational database management system (RDBMS) with cloud storage services. The RDBMS is essential for managing structured data, including game progress, user profiles, and game statistics, while cloud storage is crucial for handling media assets such as stock images. This hybrid approach ensures scalability, efficient data management, and seamless accessibility across platforms, making it the optimal choice for both performance and future growth.
4. **Memory Management**: It is highly recommended that the web-based platform leverages the automatic memory management capabilities of modern web browsers. By utilizing built-in garbage collection for memory allocation and deallocation, developers can avoid the complexities of manual memory management. This approach not only optimizes memory usage but also significantly reduces the risk of memory leaks, ensuring smoother performance and greater efficiency across the platform.
5. **Distributed Systems and Networks**: To ensure seamless communication across multiple platforms, it is strongly recommended that the game adopt a distributed software architecture and leverage network connectivity through a centralized server or cloud infrastructure. This infrastructure will serve as a critical communication hub, managing game synchronization, real-time updates, and message exchanges between players on various devices. Additionally, the system must be equipped to manage network challenges, such as intermittent outages or low bandwidth, by integrating robust error handling and synchronization mechanisms. This approach is essential for delivering a smooth and reliable user experience across all platforms.
6. **Security**: To ensure the protection of user information across platforms, it is strongly recommended that several robust security measures be implemented. The chosen web-based platform must support secure communication protocols, such as HTTPS, to encrypt data in transit. Additionally, user authentication and authorization mechanisms, like multi-factor authentication (MFA) alongside traditional username and password systems, should be employed to control access to game features and user profiles. Furthermore, data encryption techniques must be applied to safeguard sensitive information, both at rest in databases and during transmission across the network. These measures are essential for maintaining user trust and complying with security standards.