

Draw It or Lose It Web Application

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/25/2025 | Quang Nguyen | Initial draft of the software design document, including the executive summary, design constraints, and domain model |

**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](#_heading=h.35nkun2)

The Gaming Room has asked Creative Technology Solutions to create a web based version of their game, *Draw It or Lose It.* Currently the game is only on Android and the goal is to make it available on more platforms, like web browsers and mobile devices, which would allow the Gaming Room to reach a larger audience and increase player engagement.

To achieve this, we will design a web-based version of the game using efficient software design principles. By using patterns such as the Singleton design pattern, we will ensure that only one instance of the game runs at a time, avoiding potential conflicts and ensuring smooth gameplay. we will also use object-oriented programming principles to build a modular and scalable system that is easy to maintain and update as The Gaming Room’s needs grow

the solution will focus on:

* Supporting multiple teams and players
* Creating unique identifiers for games, teams, and players.
* Ensuring security and scalability.
* Providing cross-platform compatibility.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_heading=h.1ksv4uv)

* **Cross-Platform Compatibility**: The game must function on web browsers and mobile devices with different operating systems. This constraint would impact development by requiring the use of web technologies, such as HTML5, CSS, and JavaScript.
* **Unique Identifiers:** Every game, team, and player must have a unique identifier to avoid any conflicts. this requires careful database design to enforce these constraints
* **Limited resources on Mobile Devices:** Since the game will be played on mobile devices the developers need to consider limited processing power and memory. Optimizing the code and graphics for performance will be important to ensure a smooth user experience.
* **Concurrency Management**:The game must support many players at the same time. The server will handle multiple processes to keep user interactions separate and running smoothly. Using a multithreaded approach will allow each player’s actions to be managed efficiently, ensuring smooth gameplay.
* **Memory Efficiency:** The system needs to use memory carefully, especially on devices with limited resources. It will only load necessary game data to reduce memory usage. Techniques like virtual memory and caching will ensure the game runs smoothly by keeping active sessions in memory.
* **Security**: User data and game sessions must be protected. This constraint involves encrypting sensitive information and implementing authentication to prevent unauthorized access.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

The UML class diagram shows the relationships between the main components: Entity, Game, Team, Player, and GameService.

* **Entity Class:** The Entity class serves as a base class for Game, Team, and Player. It contains shared attributes, such as id and name, which ensure that all derived classes have a unique identifier and name. This demonstrates inheritance, as common properties and methods are centralized in a parent class which reduces redundancy.
* **Game Class**: The Game class represents individual games and contains a list of Team objects. It provides methods to add and manage teams (addTeam) and generate string representations (toString). The composition relationship between Game and Team ensures that each game manages its own set of teams, keeping the data organized.
* **Team Class**: The Team class represents groups of players within a game. It maintains a list of Player objects and provides methods to add players (addPlayer). This also demonstrates composition, as teams are integral components of the game structure.
* **Player Class**: The Player class represents individual users in the game. It is directly associated with a team and includes unique attributes like id and name. This setup ensures each player is distinct and easily identifiable.
* **GameService Class**: The GameService class uses the Singleton design pattern to manage the application state and ensure that only one instance of the service exists at a time. It is responsible for creating and managing games, teams, and players. By implementing static attributes and methods, it ensures efficient handling of game data and prevents duplicate instances.
* **ProgramDriver Class**: The ProgramDriver class contains the main method which serves as the entry point for the application. It interacts with other classes to initiate and manage game functionality.

**"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](#_heading=h.z337ya)

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** | macOS supports web hosting using Apache and Nginx but is less common for large-scale deployments. It is stable and secure but has higher hardware costs. Licensing fees may apply, and scaling up may require additional configurations. | Linux is the most widely used OS for web servers due to its stability, security, and cost-effectiveness. It supports Apache, Nginx, and other server tools. Open-source and free to use, making it ideal for scalability. | Windows Server supports IIS (Internet Information Services) for hosting. It integrates well with Microsoft services but has licensing costs. It is user-friendly but less common for large-scale deployments compared to Linux. | Mobile devices are not designed to host web applications. Instead, they act as clients connecting to the server. The app must be optimized for mobile data usage and real-time interactions. |
| **Client Side** | macOS users will access the game through a web browser which means requiring compatibility with Safari and other major browsers. Developing a macOS-specific app may require Swift, which increases development costs. | Linux users will access the game through browsers like Firefox or Chromium. Minimal additional development is needed since Linux users rely on web-based applications. | Windows users will access the game through browsers like Chrome, Edge, or Firefox. Ensuring smooth performance on Windows is essential due to its large user base. Compatibility with various screen resolutions is necessary. | The game must be optimized for iOS and Android devices. A responsive web design or a cross-platform mobile app using React Native or Flutter can help support both platforms. |
| **Development Tools** | Development on macOS can be done using Xcode, Visual Studio Code, or JetBrains IDEs. macOS supports web development with languages like Swift, Python, and JavaScript. Licensing for some tools may apply. | Linux development can be done using open-source tools like Eclipse, Visual Studio Code, and Vim. Web development is supported by Python, PHP, JavaScript, and Java. Free and open-source tools make it cost-effective. | Windows supports development through Visual Studio, JetBrains IDEs, and Eclipse. It works well with .NET, C#, JavaScript, and Python. Some IDEs and tools may require licensing fees. | Mobile development requires Android Studio (for Android) and Xcode (for iOS). Cross-platform development can be done with Flutter, React Native, or Unity. Some tools may require licensing for commercial use. |

## 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:** I recommend using a LInux based server to expand Draw it or Lose it to multiple platforms. Linux supporters scalable deployments and provides strong security features. Linux is also stable, cost effective and is often used for hosting web applications. Linux also has lower operational costs compared to proprietary systems.
2. **Operating Systems Architectures**: The game will use a client-server architecture where the server handles game logic data storage and user authentication and the client (Players) interact with the game through web browsers. Having an architecture like this provides centralized control which makes update and maintenance a lot more manageable. Also by separating the client and server responsibilities the game can support multiple platforms like Windows, macOS, Linux, iOS, and Android.
3. **Storage Management**: A cloud-based storage like AWS S3 or Google Cloud Storage would be the best for storing game assets, user data, and session data. Cloud storage ensures high availability, reliability, and scalability which allows the system to handle increasing player traffic without affecting performance.
4. **Memory Management**: To ensure the game runs smoothly and efficiently the system will use caching with tools like Redis to store frequently used data. This will reduce the need for repeating database queries which would improve load times. Also using virtual memory will ensure that only active game sessions and player data are loaded into memory, which would minimize resource consumptions.
5. **Distributed Systems and Networks**: To keep the game running smoothly without any crashes or slowdowns, multiple servers will be used to spread out the player traffic. Also using Load balancers to distribute the incoming player traffic across the multiple servers which will prevent any of the servers from overloading. Websockets will be used for real time updates so players can see changes instantly like scores or game events.
6. **Security**: Security is important to protect player data game integrity. HTTPS encryption will be used to secure communication between clients and the server, this protects sensitive player data from potential cyber threats. The game will also use secure login methods like OAuth2 authentication and multi factor authentication to ensure secure user logins and prevent unauthorized access. Also data integrity measures like role based access control and database encryption will also guard player information from breaches.