

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 | 11-16-2024 | Brandon Coleman | Initial draft |
| 1.1 | 12-15-2024 | Brandon Coleman | Improved 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 Gaming Room seeks to expand its popular Android game, *Draw It or Lose It*, into a web-based application that can serve multiple platforms. This expansion aims to reach a broader audience by making the game accessible on various devices.

To achieve this goal, we propose developing a scalable, web-based version of *Draw It or Lose It*. The solution involves creating a robust application architecture that supports multiple teams and players and maintains a single instance of the game in memory at any given time.

## Requirements

* Support for multiple teams
  + Game must have the capability to involve one or more teams
* Multiple players per team
  + Each team must support multiple players
* Unique game and team names
  + All game and team names must be unique within the system
* Single game instance in memory
  + Only one instance of the game should exist at any given time

## [Design Constraints](#_2et92p0)

* Web based, distributed environment
  + Application needs to work efficiently over the Internet with the challenges of latency and network instability
* Unique game and team names
  + All game and team names must be unique to prevent conflicts and maintain clear identification
* Single game instance in memory
  + Only a single instance of the game should exist at any given time
* Platform diversity
  + The application should provide the same user experience and user interface regardless of device or operating system
* Security
  + Application must protect user data and interactions via encryption using HTTPS
* Real-time interaction
  + The game relies on real-time rendering of images and user interactions

## [System Architecture View](#_ilbxbyevv6b6)

* Client layer
  + Uses web technologies to provide same user experience and interface for all devices and operating systems
* Server layer
  + Handles authentication, game logic, and persistence interaction layers
* Database layer
  + Stores the user information, game states, team/player data, and other required data
* Other layers may arise as development planning occurs

## [Domain Model](#_8h2ehzxfam4o)

The provided UML diagram outlines the structure of *The Gaming Room* system using object-oriented design principles. At its core, the *Entity* class serves as a base for *Game*, *Team*, and *Player*, encapsulating shared attributes (*id* and *name*) and methods. This promotes inheritance and code reuse. *Game* contains a list of *Team* objects, and each *Team* holds a list of *Player* objects, demonstrating composition to model hierarchical relationships.

The *GameService* class, using the Singleton pattern, ensures only one instance manages the overall system, fulfilling the requirement of maintaining a single game instance in memory. It also generates unique IDs for games, teams, and players through static variables. The diagram supports encapsulation by keeping attributes private and accessing them via public methods, while polymorphism is evident in the overridden *toString()* method in each subclass. These principles collectively support the software requirements, including managing multiple teams and players, ensuring unique names, and maintaining an efficient and extensible design.

**"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** | While Mac can be used, hardware is expensive and have limited support for web hosting. | Hosts majority of web applications with distributions that are well supported and very secure. | Great support for hosting a .NET application but licensing is expensive and more supported for large enterprise. | Not used for web hosting. Standard operating systems do not support hosting and would require a bespoke solution. |
| **Client Side** | Web browsers provide a uniform experience that only changes by browser manufacturer or screen size. Can support screens both small and large. | Web browsers provide a uniform experience that only changes by browser manufacturer or screen size. Can support screens both small and large. Would need a distribution that supports a graphical interface. | Web browsers provide a uniform experience that only changes by browser manufacturer or screen size. Can support screens both small and large. | Web browsers provide a uniform experience that only changes by browser manufacturer or screen size. Typically on the smaller end of screen size and power so interface may be simplified for size or power constraints. |
| **Development Tools** | Supports a variety of languages and editors for client and server technologies. Provides access to a Unix shell and has a robust selection of third party tooling. Vscode is popular for editing and JavaScript is popular for client and server development. | Supports a variety of languages and editors for client and server technologies. Provides access to a shell. Third party tooling is limited. Vscode is popular for editing and JavaScript is popular for client and server development. | Supports a variety of languages and editors for client and server technologies. Provides access to Linux via WSL. Excellent support for .NET development. Vscode is popular for editing and JavaScript is popular for client and server development. | NA. Being a web application, we would not develop the software to be deployed on mobile devices. |

**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**: Recommended to use a distribution of Linux such as Ubuntu for its security, open-source codebase, and popularity for web hosting. Ubuntu, in particular, is popular for its ease of use, robust support for web technologies, and large community support. Its stability and lightweight resource usage make it ideal for hosting server-side applications, ensuring smooth operation as user traffic grows.
2. **Operating Systems Architectures**: Optimal distributions such as Ubuntu provide support for popular web hosting technologies and server language technologies. System libraries provide support for popular technologies like **NGINX**, **Apache**, and server-side scripting languages such as **Node.js** and **Python**. The simplified user interface layer, often minimized on servers, reduces overhead, ensuring the platform focuses solely on performance and resource optimization.
3. **Storage Management**: Cloud storage with an S3 compatible provider offers scalable, reliable, and secure storage for blob data. Database servers such as PostgreSQL can cover the non-blob data in a similar fashion. For structured data, it is ideal as it efficiently handles relational data, such as team and player information. This approach ensures scalability, reliability, and cost-efficiency. Cloud storage also offers automatic backups, fault tolerance, and rapid data retrieval, ensuring the game remains responsive and resilient as user demands increase.
4. **Memory Management**: Cloud computing allows for scalable solutions to alleviate memory constraints. Memory intensive tasks can be offloaded to background job servers or other solutions if memory usage gets too high. Memory-intensive tasks, such as game state calculations or heavy database queries, can be offloaded to background workers or job queues, reducing the load on the main application. This approach enhances the game’s performance, allowing Draw It or Lose It to maintain real-time interactions seamlessly.
5. **Distributed Systems and Networks**: Websockets provide a more real-time solution. Solutions for distributed systems would come as the need develops. Typical solutions are region locking of games, sharding of the databases, and utilizing CDNs to reduce asset latency.  For further optimization, region-specific servers can improve connectivity for geographically dispersed users, enhancing overall gameplay.
6. **Security**: Data can be encrypted at rest, access strictly limited to servers, and strict use of HTTPs can lessen security problems. Servers can be protected at various levels such as local firewall, IP filtering, and DDoS protections. By implementing these measures, the system meets modern security standards, ensuring user trust and data integrity.