

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 4**

[**Domain Model 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/22/25 | Brighton Ulery | <Brief description of changes in this revision> |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is an Android game application developed by The Gaming Room. Based loosely on the 1980s television series, *Win, Lose or Draw*, teams of players take turns guessing what a progressively revealed image depicts. Each game consists of four rounds lasting one minute each, with the image being fully revealed at the 30-second mark. To expand their player base and improve accessibility, The Gaming Room would like to develop a cross-platform, web-based version of their Draw It or Lose It game.

## Requirements

* Business Requirements
  + Include team-based multiplayer support.
  + Ensure uniqueness through unique game, team, and player names and identifiers.
  + Select and progressively reveal a randomly chosen stock image.
  + Consistent timer, team priority, and round changes.
  + Ensure only one instance of the game service exists at a time.
  + Provide scoring and win conditions.
  + Provide a user-friendly interface to attract users.
* Technical Requirements
  + Develop a web-based, cross-platform framework.
  + Integrate a database across their website and mobile application.
  + Provide session and state management.
  + Provide a drawing rendering system.
  + Ensure scalability for increase in userbase.

## [Design Constraints](#_2et92p0)

1. UUID/Unique Naming System

The system must provide a system that verifies team and player name uniqueness as well as universally unique identifier (UUID) assignment to games, teams, and players.

1. Single Active Game Instance

The system must only have a single instance of the game service in memory at any one time.

1. Web-Based, Cross-Platform Framework

The system must support and be consistent across different browsers and devices.

1. Cross-Platform Database Integration

The new system’s database must be unified between the web-based and Android versions.

1. Real-Time Synchronization

The system must synchronize the timers, team/player turns, and progression of the reveal of the image across all users.

1. Session and State Management

The system must track games, teams, and users as well as game aspects such as number of rounds and the current score across sessions.

1. Scalability

The system must be developed dynamically to ensure that as the user base increases, the system can be built upon to withstand the increased load.

## [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)

Below is the working UML design for the system. The superclass (parent) Entity contains generalized attributes, the id and name, which are encapsulated to prevent unintentional modification by external sources, as well as helper methods for retrieving these values and displaying object information. The subclasses (children) Game, Team, and Player, through the application of inheritance, share these characteristics from the parent class. The GameService class is designed using the singleton pattern, ensuring that only a single instance of the GameService will exist in memory at a time. Through composition, the GameService class maintains a list of 0 or more Games, which in turn maintains a list of 0 or more Teams, which likewise maintains a list of 0 or more Players.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS is not designed well for hosting servers, offering very little native tools for server management. Apple officially discontinued their OS that offered support for server hosting, Mac OS X Server, in 2022, electing to include minor elements of the discontinued service into their standard macOS package. macOS can be used to host small, local developmental servers, but should not be used in production due to its lack of scalability, server-specific support, and high hardware cost. | Linux based systems are excellent for server hosting. Most models, ranging from small business IT to cloud services, use some flavor of Linux to manage their servers. Linux is cost effective, lightweight and opensource. Linux benefits from server hosting tools and APIs developed for it by dedicated and freelance developers. As Linux servers are typically run with no graphical interface, there is a learning curve to the commands and navigation. | Microsoft provides Windows Server, a branch of Windows developed with server management in mind. With powerful native tools developed by a dedicated team, Microsoft provides excellent support and documentation. Windows Server also has a graphical interface, lowering the barrier of entry. However, Windows Server does require a license to operate, which can be costly as the needs scale. | Android and iOS are not developed to support servers at all, and as such cannot be used for hosting. Both systems are client systems developed for mobile devices, as such they would be better fits for the client side rather than server side systems. |
| **Client Side** | macOS supports most major web browsers, as well as Apple’s proprietary Safari browser. Some additional time and expertise may be required when developing for Safari, as it has its own rendering system, WebKit. Developers will also need to be familiar with macOS, as testing for Safari will need to be done on a Mac machine. | Linux supports most major web browsers. There are no additional costs that would be required to develop for Linux, as most distributions are free and opensource. Some additional time and expertise may be required as the system would need to be tested across different flavors of Linux to ensure compatibility. | Windows supports most major web browsers. Windows is also one of the most widely used OS, meaning most users will be familiar with the system and how to develop with it, requiring less time and training. Developers familiar with HTML, CSS, and JavaScript will be well supported by working with a Windows environment. | Android and iOS have apps for most major web browsers; however, iOS does also mandate Safari. iOS requires a Mac system to test on their mobile devices, which is a cost factor. Additional time is required to test multiple devices and screen sizes, which is another factor in cost. Some IDEs may provide device simulation to mitigate this. |
| **Development Tools** | Relevant programming and scripting languages for client side include HTML, CSS, and JavaScript. TypeScript may also be used. Python, Java, and JavaScript are also used for server-based development. A plethora of free and paid IDEs are available. Browser development tools are also provided, with Safari’s Web Inspector being essential for developing a web-based application for Safari. | Relevant programming and scripting languages for client side include HTML, CSS, and JavaScript. TypeScript may also be used. Python, PHP, Ruby, Java, and JavaScript are also used for server-based development. Many free and open-source IDEs are available for the OS, with Vim being a popular option. Linux also offers support for more mainstream IDEs like Visual Studio Code. | Relevant programming and scripting languages for client side include HTML, CSS, and JavaScript. TypeScript may also be used. Python, Java, and JavaScript are also used for server-based development. Windows provides Visual Studio and Visual Studio Code as official IDEs supported by Microsoft. | Relevant programming and scripting languages for client side include HTML, CSS, and JavaScript. TypeScript may also be used. Swift and its derivatives are required to build iOS apps, while Android apps require Java and its derivatives. Apple’s official IDE Xcode is required for development and testing iOS apps, and a Mac is required to test on mobile devices. Android apps are typically developed using Android Studio. |

Recommendations

1. **Operating Platform**: Based on our evaluations, we recommend a Linux-based server platform due to its cost-effectiveness, flexibility, and scalability. With a wide range of open-source and professional server management tools, a Linux-based server platform is equipped for efficient deployment and monitoring. Linux-based server platforms are the most popular system for server deployment and are an ideal fit for the server-side of *Draw It or Lose It*.
2. **Operating Systems Architectures**: Linux-based platforms have a monolithic kernel architecture, meaning each of its core functions and services operate in a single kernel process. This architecture ensures speed and efficiency when components are communicating with each other. Linux-based platforms also support multitasking and concurrent network connections, enabling it to handle multiple game sessions and player connections without experiencing loss in performance. Another key feature of Linux is its modularity; required services and functionality can be kept on while unnecessary features can be disabled to improve efficiency.
3. **Storage Management**: For storage, we recommend using Google Cloud Storage (GCS), an affordable cloud-based system, for general program files such as the pre-generated images. GCS integrates well with Linux, providing command line interface (CLI) tools for ease of access. We also recommend PostgreSQL, a database management system, for storage of structured data sets, such as players, teams, and game sessions. PostgreSQL is optimized for Linux, providing CLI tools for management and optimization. Both systems are also scalable, perfect for a game seeking growth like *Draw It or Lose It*.
4. **Memory Management**: Linux-based systems use paging, incorporated into a virtual memory management system, allowing it to allocate memory over multiple active processes at a time. Inactive or unused processes have their memory usage reduced or are freed. This ensures that memory is available to services that need it. Linux also incorporates caching, reducing the memory load of segments of the game that are called upon often.
5. **Distributed Systems and Networks**: *Draw It or Lose It* uses a client-server model. Clients on different platforms can communicate with the server using RESTful API calls, enabling them to interact seamlessly. To ensure low-latency and reduce the chance of outages, redundancy, connection monitoring, and timeout handling should be included. Fallback systems such as server clusters should be implemented in the event of unexpected network failures.
6. **Security**: In terms of security, all communication between the client and server should be done over HTTPS with TLS. This ensures that data is encrypted between endpoints. Linux-based systems also include a number of security features for protecting the server, including firewalls, user/group permissions, and process isolation tools. This ensures unauthorized access to the server is prevented and protects information stored within it.