

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 8**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 06/22/2025 | Mark Quackenbush | Filled in the Recommendations section of the Template |
| 2.0 | 06/08/2025 | Mark Quackenbush | Filled in the Evaluation section of the Template |
| 1.0 | 05/26/25 | Mark Quackenbush | First Draft of Software Design Template |

**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 would like to expand the functionality of their game “Draw It or Lose It” to be operational on multiple platforms, to extend its audience reach past its current sole availability on the Android app store. This will involve developing a web-based version of the game and properly setting up the environment for it to be hosted on, while meeting all necessary software requirements as specified by The Gaming Room.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_2et92p0)

* A singleton design pattern should be used to ensure that no more than one instance of the game can exist in memory at any given time.
* The interface of the game must be accessible via any device that is able to connect to the web.
* Unique identifiers should be used for each game, team, and player names to prevent multiple occurrences within the system.
* Each round of the game will last one minute, and each game will consist of 4 rounds.
* The images to be guessed by players in a game will be derived from a library of stock drawings that are randomly selected. Each drawing will slowly be drawn out over the course of a 30-second interval, during which time the player will have the opportunity to guess what is being drawn.
* If the player team is unable to guess the drawing in the 30-second timeframe, the remaining teams will be given a 15-second interval to guess the drawing in order to “steal” the points.

## [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)

Inheritance is heavily used in the Domain Model for the Draw It or Lose It application, by using the Entity class as an abstract class that the Game, Team, and Player classes all inherit from. That is to say, the Entity class can not be instantiated itself (as an instance of an “Entity” would be vague and unhelpful), but all attributes and methods defined in the Entity class must be included in the definitions of its subclasses. This allows each Game, Team, and Player object to be instantiated as unique objects within the program.

The Model also defines a stream of “0 to many” relationships between classes in the program: the GameService class can only have one instance at a time (due to the singleton design pattern detailed elsewhere) but can host multiple instances of games within the service; within one instance of the Game class, there can be many instances of Team objects; and within one instance of the Team class, there can be many instances of Player objects.

The singleton design pattern is utilized by the GameService class to ensure that only one instance of the GameService object can be in use at any given time. This is enforced by defining the class’s own service attribute with the value of a specified GameService object and making the class’s constructor a private method so that additional instances of the class can not be created by other classes. This is an example of encapsulation as an OOP principle in the Model’s 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** | - Utilizes a built-in Apache web server with HTTP support.  - Many people already own a Mac and are familiar with the interface.  - Utilizes similar levels of stability and security with Linux systems (due to being Unix-based).  - Certain Mac products can be equipped with very high computing power.  - Significant limitations in adaptability and scalability for a server environment.  - Very little long term support, as new rollouts often make previous versions obsolete.  - Extremely high costs compared to similar Linux environments. | - Dominant choice for web hosting due to being open-source, reliable, and cost-efficient.  - Highly scalable and adaptable.  - Systems can run for incredibly long uptimes without reboot, because most patches don’t require a system reset.  - Main challenges come from the CLI learning curve and system expertise necessary.  - Due to being open source, there is no official “vendor” support for most Linux flavors, so troubleshooting often relies on community support. | - Most people are familiar with navigating a Windows environment, and the user-friendly GUI makes for a fairly easy learning curve.  - Multitude of user-friendly “Wizard” tools to handle much of the grunt work, and seamless compatibility with many pre-built Windows services, as well as robust official vendor support.  - Main challenges deal with high cost licensing fees and higher resource consumption.  - Windows also frequently patches their system, requiring downtime on the server to install updates. | **iOS**  -iOS is a closed platform that runs on battery-powered devices. It is not designed to be used a server hosting platform.  -iOS also typically does not allow for running in virtual machines.  **Android**  -Android devices allow comparatively more flexibility than iOS, but are generally not regarded as a viable option for running a web server.  - They lack the robust hardware necessary to run a large-scale web service, and the battery constraints of mobile devices make them impractical for long-term usage. |
| **Client Side** | -For a native Mac app, languages like Swift and Objective-C are typically used to program. Though they work very well with a Mac environment, they are fairly niche languages, and might require specialized teams to deploy a Mac specific application.  - MacOS is also required for developing iOS client apps, as they must use XCode to program the client app, so they can be very effective as a development environment. | -Linux’s robust and varied usage means that it has capabilities to properly support almost any programming language, but popular uses for game servers specifically include JavaScript/TypeScript, Python, and Ruby.  - Very few gaming applications have Linux native apps, but using a web browser, a gaming service can be accessed through a web-hosted game.  - The well built-out and supported nature of Linux, means that it should be relatively low-cost to develop and finding the necessary expertise should not be prohibitive. | -To develop a Windows native application, C++ with DirectX/Vulkan is often used, as well as Unity (for games specifically). These are commonly used in development, so it likely would not be too hard to find specialized developers for those tools.  - Windows also provides many robust tools (like Visual Studio and WSL) for development, that may keep costs within a reasonable budget. | **iOS**  -iOS client development usually utilizes Swift and Objective-C, similarly to Mac devices.  -Apple’s App Store is notoriously strict, and has the ability to limit what apps can be vended on the store, making client development subject to regulatory limitations.  **Android**  -Client development is usually done with Java, Kotlin, and occasionally C++. Many developers are well-versed in these languages, so finding the necessary experts should be fairly easy.  - Because many different manufacturers utilize an Android framework, there may be device-specific quirks, making development more complicated and potentially more costly. |
| **Development Tools** | **Programming Languages**  -Node, Python, Swift for server-side.  -HTML/JS (Web app) and Swift/Objective-C (Native Mac apps) for client-side.  **Dev Tools**  -Terminal + Homebrew, Xcode as a general IDE, and Docker Desktop for running Linux containers on the server-side.  -Safari Web Inspector for web app debugging, and Xcode as an IDE for client-side. | **Programming Languages**  -Node.js, Python, Java, Go, Rust, C++, PHP, .NET core for server-side.  - HTML, CSS, JavaScript/TypeScript, WebAssembly for client-side (all web-based, as there are typically no native Linux gaming apps).  **Dev Tools**  -Apache/Nginx servers, MySQL/Postgres databases, Docker/Kubernetes, GCC/Clang compilers, Shell scripting, and various open-source libraries for server-side.  -Chrome/Firefox browser for accessing the web application client-side. (JS/HTML for the web-based development). | **Programming Languages**  -C#/.NET, C++ for legacy code, Node.js, Java, Powershell for scripting, for server-side.  -HTML/JS for web-based app, C# or C++ for Windows native app, for client-side.  **Dev Tools**  -IIS web server, Visual Studio IDE, SQL server or Azure integration, Windows Server management tools, and potentially Docker for server-side.  -Chrome/Edge/Firefox dev tools for browser debugging, Visual Studio for native development, Xbox SDKs if integrating with Xbox, for client-side. | **Programming Languages**  -iOS: No server-side languages as mobile isn’t suited for hosting servers. Swift and Objective-C for native app development, HTML/JS for web-hosted (running on Safari/WebView) for client-side.  -Android: No server-side languages as mobile isn’t suited for hosting servers. Java or Kotlin for native apps, C++ with NDK for performance libraries, HTML/JS for web-based, for client-side.  **Dev Tools**  -iOS: Xcode IDE for app development, TestFlight for beta testing, Safari’s remote debugger for web-based.  -Android: Android Studio IDE, Logcat and Android Profiler for debugging, Unity/Unreal editors, Chrome DevTools for web-based. |

## 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**: Overall, the seemingly best Operating Platform to support the expanded needs of The Gaming Room to bring Draw It or Lose It! to multiple platforms would be a Linux system.
2. **Operating Systems Architectures**: Linux is a highly scalable option when it comes to web-hosting. It’s open source and adaptable nature makes it a strong choice for developing the application further for the varied needs of multiple platforms, due to the fact that the operating system can support development in most languages, making it well-suited for expansion to multiple platforms. Though not typically used as an environment to host a native instance of gaming apps, Linux can still access the application through web-based instances of the game. Due to there being various different flavors of Linux distributions, there is likely to be a specific version of Linux that is suited best for the needs of hosting the application, with robust community support to help troubleshoot any issues with the deployment.
3. **Storage Management**: The application requires the storage of around 200 high-definition images in non-volatile storage. Because the number of images and their storage usage requirements will not change over time (unless more images are added to the bank of existing images), the solid state storage requirements of the system are easy to anticipate and should not be expected to fluctuate over time. This means that the Gaming Room could host in-house storage to hold the various images used in the game’s functioning, until they need to be accessed by faster methods for consistent rendering.
4. **Memory Management**: Due to anticipated fluctuating demand for game usage (dependent on popularity during certain times compared to others), the RAM requirements necessary for the Gaming Room to reliably accommodate all players may vary drastically. In order to avoid overspending on memory that will only be occasionally utilized to its full extent at certain specific times, a cloud-computing infrastructure to manage the memory needs of the application is recommended to dynamically allocate memory usage dependent on client demands at any given time. Linux systems support various platforms that allow for cloud computing infrastructures, including use of reliable cloud services such as AWS and Azure to suit the application’s needs.
5. **Distributed Systems and Networks**: An easy way to allow the Draw It or Lose It application to function across multiple platforms is by hosting it on the web, so that any device that can access the internet (regardless of its operating system) is able to play the game on the web. However, the game would ideally also have native instances of the application that allow it to run on the operating system directly. Due to Linux’s robust capabilities to support most programming languages, this makes the development of client-side native applications well-suited to Linux’s system. Additionally, on the server-hosting side of the application, Linux allows patches to the system without need for full reboots, meaning the server running the game should ideally be capable of operating for extended periods of time without any need for downtime (with the exception of occasional major patches).
6. **Security**: Luckily, due to the nature of Draw It or Lose It, the app doesn’t require the intake of much personally identifiable information due to it being a fairly simple gaming app, the security needs of the Gaming Room do not necessarily need to be extremely robust. However, the application still requires the use of account log-ins to distinguish between player accounts, so the topic of security cannot be outright ignored either. For the use of log-in authentication, REST-ful APIs should be able to handle the basic needs of encryption and authorization for user accounts on the system. Linux specifically has well-established systems for security such as Linux Unified Key Setup (LUKS) to manage Full Disk Encryption for the server hosting the application, as well as network security tools such as iptables to establish firewalls to prevent bad traffic from accessing the server’s network and Secure Shell (SSH) to secure remote access communications on the Linux system.