

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 | 09/20/2025 | Miterrand Reyes | Completed template for technical progress review. |
| 2.0 | 10/04/2025 | Miterrand Reyes | Completed evaluation section of template for review. |
| 3.30 | 10/19/2025 | Miterrand Reyes | Completed recommendation section of 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 Rooms product, Draw It or Lose It, is currently an Android-exclusive gaming application, limiting its reach to a single development platform and user base. The creators aim to transition their product from the single-platform mobile environment into a cross-platform web-based application. Guiding this transition effectively is crucial for the product’s success. Expanding into multiple operating systems will allow Draw It or Lose It to reach a larger customer base and solidify future growth for The Gaming Room.

## Requirements

* *Game must be able to run on multiple web-based environments*
* *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)

* Cross-platform compatibility:
  + One of the most important goals is the ability for the game to run on multiple operating systems and their respective browsers. A challenging constraint that must be carefully navigated as each different OS brings different hardware, file systems, and browsers.
* Team-based gaming structure:
  + Game code is inherently hierarchical. The “game” must consist of a “team” which is made up of “players”. A crucial element which the design must reflect.
* Naming uniqueness must be enforced for games and teams:
  + Names for games and teams must be unique to prevent conflicts. Real-time validation must be implemented to avoid conflicts. Duplicate records are not allowed, as this will prevent unwanted conflicts that would otherwise hurt the user experience.
* Single instance of the game (Singleton design):
  + Game control must be kept to one active session in memory. Reiterating the importance of uniqueness in every aspect of the game service. Limiting multiplayer experiences but cementing the idea of preventing conflicts for the users.

## [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)

In this domain model, class relationships are made clear by the use of arrows and connecting lines. The Entity class serves as a base class that contains common attributes and methods. Game, Team, and Player all inherit from Entity and are examples of how code can be reused using inheritance. The GameService class manages Game objects and is designed using the Singleton pattern. Ensuring a single instance of the game is created each time. The Game class contains multiple Team objects, and each Team contains multiple Player objects. This trait between Game, Team, and Player forms a clear hierarchical structure that mirrors the real-world relationship of these classes.

Object-oriented principles are evident throughout the UML. Classes contain private variables that encapsulate data from each other and only expose the class's public variables and behavior through public methods. Inheritance is clearly demonstrated by using the Entity class as a superclass, which reduces redundancy in the application. Game, Team, and Player are all child classes that inherit properties and behaviors from the parent Entity class. Entity is also a perfect example of abstraction; the class uses the general attributes, id, and name, which are concepts. Something needs to be referenced with an ID and a name. Once a subclass inherits these, it becomes clear that every Game, Team, and Player needs to have an ID and a name. Finally, polymorphism is used by each subclass, Game, Team, and Player. Each class overrides the toString() method, which they inherited from Entity, to provide class-specific string output.

**"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 inormation.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS offers powerful hardware that has more than enough capability for web hosting a gaming app. Unfortunately for The Gaming Room, Mac servers aren’t recommended. macOS is a desktop-focused operating system first. This can lead to numerous compatibility issues when implementing business requirements, such as hosting a gaming app. Widely used server applications, such as Apache or Nginx, are available for macOS, but they are not optimized for the OS. Specifically, macOS would not handle a large-scale deployment efficiently due to the lack of optimization. Support for these servers is also limited because they aren’t an industry-standard server. | Linux is one of the heavyweights in the server industry standards. One thing to keep in mind is that Linux is a lightweight operating system that can be easily integrated into any server hardware option. This is an advantage as the OS offers professional command-line tools and supports many server applications. Linux highlights scalability, which will empower The Gaming Room to increase its customer base. One consideration to keep in mind is that Linux is not as user-friendly as other operating systems, and it will require technical expertise to navigate the command-line interface correctly for setting up an appropriate server. | Windows servers are another industry-standard option. Offering both powerful hardware options and a tried-and-true operating system that is widely supported by all standard server applications. Windows servers are available to meet any business need, from minor to enterprise-level. These edition-type server options do come with required licensing costs. Windows offers multiple server editions, each with its own price. Additionally, the Windows server hardware is highly customizable, so finding the perfect fit for The Gaming Room could vary significantly in terms of price, depending on the OS edition and Windows hardware. | Mobile devices are not designed to run as servers or host server applications. Lacking in many areas, such as hardware capabilities and dedicated infrastructure, such as a central network room, where they can constantly live. For The Gaming Room, mobile devices would primarily be used on the client side, rather than for hosting their cross-platform application. |
| **Client Side** | macOS has powerful hardware options for gaming. One thing to keep in mind is ensuring that the Safari browser is correctly linked to the hosting server. Apple has specific quirks, such as strict security policies, which are meant to protect the system. Login may be different in this system than in a typical Windows environment. MacOS development expertise will also be required to handle all the operating system-specific details. | The Linux operating system offers multiple distributions, each with its own graphical user interface. Distros such as Ubuntu run common web browsers like Google Chrome and Firefox effortlessly. With this foundational layer checked off, development is straightforward. Utilizing industry-standard web development techniques involving HTML or JavaScript. Linux is a flexible OS that will keep development costs down. Linux does have some distributions that don’t carry a GUI, but these are more specialized for the server side of the application. | Windows is the most widely used operating system. Development is straightforward using industry standards such as HTML or JavaScript. Although it is the newest browser, Microsoft Edge also aligns with these industry standards, allowing for smooth development on the top three Windows browsers: Edge, Chrome, and Firefox. One thing to keep in mind is that since Windows supports all these effectively, cross-browser testing will need to be done to ensure everything looks correct for the gaming application as spacing and placements might need to be handled differently across the browsers. | Mobile device gaming is something that The Gaming Room is already utilizing. To further enhance this experience, responsive web design is crucial, as it ensures a consistent look across today's ever-evolving mobile devices. A significant advantage of this approach is that once the code is written for Android, it remains consistent across that specific system. There are multiple challenges when developing for both Android and iOS, as they both have different quirks, and it must be kept in mind that these devices solely use a touch interface. A specialized group for mobile device app development may be necessary for improvements and establishing server connections. |
| **Development Tools** | macOS utilizes the free Xcode IDE, which requires knowledge of the Swift programming language, for Apple development. Java Eclipse or VS Code can also be used for development. Safari DevTools can be utilized to write the HTML or CSS that the web app will need to be developed in. Development will require Apple hardware for testing on the Safari browser. | Linux can run industry-standard IDEs such as Eclipse and VS Code. Java, JavaScript, Python, HTML, and CSS are all languages that can be used for development. All these tools are open-source, and Linux accommodates their various IDEs accordingly. A Linux distribution with a graphical user interface will be needed for testing the different browsers. | Windows supports powerful IDEs such as Visual Studio, Eclipse, VS Code, and many more. Being the most widely used OS, all these are supported, and the respective browsers on which the web app will run are all available. These tools are all available for free use. | Mobile devices have IDE tools such as VS Code, Android Studio, and Xcode. Android uses languages such as Kotlin, Java, and React Native. iOS uses languages such as Swift, Objective-C, and React Native.  Will require physical mobile devices or emulators for testing the application on all the different mobile device browsers. |

## 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**:

Achieving expandability requires building a backend capable of supporting the diverse gaming platforms used today. A solid foundation that does not sacrifice performance or user experience is vital to achieving expandability. For this reason, a Windows Server hosted within Microsoft Azure is the recommended operating platform. First and foremost, Windows Server offers robust enterprise-level support but keeps it’s familiar interface known across the world. Providing industry-standard server applications and development tools, Windows Servers offer a scalable and secure solution. Both of these benefits are elevated without having to rely on physical metal boxes for Windows Servers. Microsoft offers foundational headless solutions through its dedicated cloud service, Microsoft Azure. Utilizing a virtual infrastructure, Windows Servers are capable of providing diverse options that come with flexible licensing options to match the company’s budget.

1. **Operating Systems Architectures**:

Windows Servers operate on a hybrid kernel architecture. The kernel is the core part of an operating system. In practice, servers can run as a monolithic kernel, where everything runs together, making operations fast, but they could suffer from crashes if one part of the system has a bug. Servers can also run as microkernels, where only the most essential operations run in kernel mode, and everything else runs separately in a user mode, making the system safer but a bit slower. Windows servers' hybrid kernel architecture takes both these concepts and combines the speed of monolithic server operations with the modular separation of microkernels. Core server operations run faster while the drivers that would process images are separated. Separating background processes, such as image drivers, minimizes the chance of complete system crashes due to one of these processes failing. In summary, Windows Servers create a clear distinction between user mode and kernel mode, preventing application-level issues from disturbing the operating system’s core.

1. **Storage Management**:

The Gaming Rooms product Draw It or Lose It will start with 200 high-definition image files, each around 8 MB in size. But by no means does this need to be a limiting number; Windows Server offers an efficient and scalable storage solution that can be utilized. Going serverless allows The Gaming Room to use Microsoft Azure’s vast cloud storage solutions, which encompass more than just image storage but also include user information storage.

* **Azure Blob Storage** – Storage solution for creative works has key features like encryption and the ability to integrate into a Content Delivery Network, which puts these large files at dedicated endpoints closer to users across the world.
* **Azure AQL Database** – A relational database service that supports SQL syntax. Scalable and structured, perfect for storing user account information, player statistics, and multiple chart style gaming information.
* **Azure Archive** – Long-term storage solution for gaming application log backups.

Scalability is the keyword here, speaking on the primary storage solution, Azure Blob Storage can increase to accept The Gaming Rooms' growing image database. Additionally, these solutions are part of the entire Azure structure, which allows for seamless interaction with the gaming servers that will host the application.

1. **Memory Management**:

Windows Server on Azure uses virtual memory management. Efficient concepts are utilized so the virtual server can run optimized and stable. Paging, segmentation, and dynamic allocation all come together so the operating system stays on top and also gives the developer power for memory use. On top of this, Microsoft Azure utilizes robust monitoring tools, Azure Monitor, which tracks the performance of the system and can trigger alerts if inefficiency is discovered.

* **Paging(OS)** – Paging breaks memory into blocks called pages. Only the pages needed are loaded, improving efficiency even when memory is limited. Automatically running behind the scenes.
* **Segmentation(OS)** – Divides memory based on logical sections of the program. Separating code logic from user input and organizing memory into sections that make sense improves stability.
* **Dynamic Allocation(Dev)** – Allows developers to allocate and release memory in the code. Giving control over short-lived tasks, such as score updates. A vital concept for keeping the application running fast.

A simple memory management workflow would be triggered by a user input to initiate a gaming session. The virtual server would begin to run the code to process the images to be sent to the client's memory. At this point, the memory would be dynamically allocated for image handling, then released when the image has been processed. Within the image processing, the image would be retrieved from Azure Blob Storage. Once done, processing the image will be sent back to the client utilizing the OS level memory concepts. Azure monitors all of this, so the entire process can be fine-tuned if need be.

1. **Distributed Systems and Networks**:

A serverless backend is the optimal solution for distributing the foundation of the application. To support cross-platform communications, RESTful APIs must be used. Microsoft Azure can host these on Azure App Service. Allowing users to consume the APIs by any client operating system as RESTful APIs is a uniform communication bridge between clients and servers using standard HTTP requests.

Combining this with Azure Blob Storage built as a Content Delivery Network allows users to quickly and efficiently receive images on a global perspective. CDNs are meant to keep information as close to the user as possible, reducing latency between the server and the user, since a single server won't always be pinged for information. Serverless instances can also be automatically rerouted to another edge node if one fails. Additionally, Microsoft Azure utilizes a robust load balancer, so all the backend instances don’t get overburdened by user interactions, and also reroutes based on load.

1. **Security**:

Security is the most critical detail of the system. If users create profiles, their information must be safely stored. Game logic and creative works are vital elements that need to be guarded against bad actors who could aim to manipulate or hack the application. Microsoft Azure keeps all the essential information encrypted; both storage and databases are protected this way. Using RESTful APIs safeguards client-to-server and server-to-client data through HTTPS.

Microsoft Azure uses Azure Active Directory, which manages user identity by using multi-factor authentication. AAD enforces role-based access control, separating users from admins.

Threat detection is constantly monitored using Microsoft Defender, Windows' built-in anti-virus software. Monitor is also built in through the Azure Security Center to prevent any threat from slipping through the cracks and be compliant with today's standards.

Microsoft Azure provides all these services on top of each other, building a robust stack of an OAuth server, firewall, backend service provider, and storage solution. All virtualized and capable of expanding horizontally to support as many users as required.