

Draw It or Lose It

# **CS 230 Project Software Design Document**

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 | 06/01/25 | Rachael Baker | Added development requirements for different  operating systems. |
| 1.1 | 06/20/25 | Rachael Baker | Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation |

## [Executive Summary](#_sbfa50wo7nsh)

The staff at The Gaming Room requires a development environment to be set up for their web-based gaming app, based on their current Android-only app, "Draw It or Lose It." This modern adaptation of the classic game show "Win, Lose or Draw" allows teams to guess puzzles based on clues derived from a library of stock drawings, rather than having a player draw them. Each game consists of four one-minute rounds, with the drawings completed at the 30-second mark. If teams cannot guess the puzzle in time, other teams have the opportunity to submit a guess within a 15-second time limit.

## Requirements

* The game will serve multiple clients and platforms.
* Each game will have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game, team, and player 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.

## [Design Constraints](#_2et92p0)

* Teams may need to be limited to a certain number of players, which may cause performance issues.
* Ensuring data integrity with concurrent users introduces complexity and potential performance hits.
* To ensure consistent image quality and game performance, using larger images will be necessary and may require additional memory and storage.
* Having games, teams, and players requires an administrator to manage them, adding to the complexity.
* The game will require additional security and authentication methods to ensure users are legitimate and user data is kept safe.
* Restrictions are needed to ensure a single user is only logged in on one device at a time.

## [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)

The UML class diagram comprises several key classes that illustrate their relationships and effectively fulfill the software requirements. The parent Entity class, which serves as a base class for others to inherit from, contains common attributes such as ‘id’ and ‘name’ for all entities (games, teams, and players), ensuring each class has a unique identifier and name. Utilizing the iterator pattern allows for checking names to prevent duplicate team or player names, and each player can be assigned to only one team at a time (Lavieri, 2019). This promotes encapsulation by restricting access to the class’s internal workings while providing a way to access information. Additionally, the singleton pattern adds another layer of game control, restricting the number of instances of the game to ensure that only one game exists in memory at a time. It does this by declaring a private constructor and instantiating a single instance inside the class, then adding a method for the program to access the single instance (Lavieri, 2019). Lastly, the program driver class runs the program, and the singleton tester class is used to test the objects and the singleton pattern.**"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 a stable and secure platform suitable for developing web-based applications. It is built on Unix, which offers good support for development tools and programming languages(Casamento & Grimshaw, 2017). Its hybrid kernel helps manage resources efficiently, making it reliable for building distributed applications (Yeung, 2018). The system supports virtualization tools like Parallels, which are useful for testing different environments. In terms of security, macOS provides user access controls, encryption, and protections like Gatekeeper, helping keep data safe. However, its licensing costs are higher, and it is not as scalable as Linux for hosting large web applications. Linux servers are usually preferred for deployment because they are open source, cost less, and work well with cloud services. | Linux is a good choice for developing and deploying the gaming application because it is open-source and based on UNIX (Bethencourt, 2016). Its monolithic kernel allows for customization, which helps during development (Bethencourt, 2016). Since Linux is free, it’s cost-effective, especially for setting up multiple environments. It also supports many virtualization tools like Docker and KVM, which help create isolated environments for testing distributed applications. Linux offers strong security features as well, such as user access controls, SELinux, and LDAP support, which help keep the application secure (Bethencourt, 2016). Setting up a Linux environment is straightforward, with many tools available for managing servers and deployment, including cloud services like AWS and Google Cloud. | Windows is a solid choice for developing and deploying ‘draw it or lose it’ on the server side. It offers a stable, hybrid kernel and supports virtualization via Hyper-V, perfect for testing multiple OS environments (Mattwojo, 2016). Security is strong with user controls and Windows Defender. Windows Server supports LDAP and Active Directory, aiding user management and security (Mattwojo, 2016). It has extensive support, documentation, and works well with many tools and enterprise apps. Plus, it integrates smoothly with cloud services like Microsoft Azure, offering scalable deployment (Mattwojo, 2016). However, Licensing is proprietary, requiring purchase for each server, which can be costly compared to open-source options like Linux. | Mobile platforms such as Android and iOS are primarily designed for end-user devices rather than development servers (Silbershcatz, 2008). Therefore, they are not suitable as hosting environments. Android, being open source, offers some support for development tools and virtualization through Android Studio and emulators, but these are limited compared to desktop OS environments (Lee, 2016). iOS is closed source and tightly integrated with Apple hardware, rendering it impractical for server-side development or hosting (mac). |
| Client Side | When supporting multiple clients on macOS, it is essential to consider the predominant browsers used on Mac, such as Safari, Chrome, Firefox, and Edge, and their compatibility with the application. Additionally, understanding that many Mac users also utilize mobile devices, including iPhones and iPads, influences the deployment strategy, emphasizing the importance of a responsive or cross-platform web application (Yeung, 2018). Development costs and time will be affected by the need to ensure compatibility across these browsers and devices, requiring expertise in web standards and extensive testing. The popularity of Mac, though less than Windows, still represents a significant user base, which justifies investment in optimized browser support and device adaptability (Yeung, 2018). Ongoing maintenance expenses include keeping up with browser updates, security patches, and ensuring seamless user experience across Mac devices and browsers (Casamento & Grimshaw, 2017). | Linux accounts for only about 2-3% of desktop OS use, with Chrome and Firefox being the main browsers, especially on Linux (Yeung, 2018). While support for these browsers is generally strong, performance and compatibility can vary across different Linux distributions (Bethencourt, 2016). This small market share means development teams might prioritize other platforms, but the rise of mobile devices underscores the need for cross-platform support. Testing across various distributions and browser versions is essential to ensure consistent functionality (Bethencourt, 2016). This process can be resource-intensive, potentially extending release timelines and affecting agility. Despite the limited user base, supporting Linux browsers is crucial for broad accessibility, especially as users increasingly rely on mobile devices and cross-platform solutions. | To support multiple clients for the Draw It or Lose It game on Windows, several software development factors must be considered. Cost involves investing in skilled developers for cross-platform and browser-based application development, along with budgeting for maintenance and licensing fees. Development timelines may be extended due to the need for compatibility across different Windows versions and the varying behaviors of popular browsers like Chrome, Firefox, and Edge. Expertise in web technologies and responsive design is crucial to ensure a seamless user experience (Mattwojo, 2024). Additionally, targeting the most widely used browsers is essential, as Windows holds a significant install base, making optimization for these platforms critical for maximizing user engagement and reach (Mattwojo, 2024). | Costs can increase due to the need for different coding approaches for iOS and Android. Maintenance demands resources for updates across all platforms. Time is a factor as catering to various platforms can extend development and testing cycles. The expertise required is significant, leading to higher hiring or training costs. Due to its increasing popularity and eliminating the need for personal computers it should be prioritized as the main client. |
| Development Tools | Programming languages include JavaScript, HTML, and CSS for front-end development, and potentially Python, Ruby, or Node.js for back-end development (Yeung, 2018). Tools and IDEs commonly used in this process may include Visual Studio Code, Sublime Text, or WebStorm for coding, along with version control systems like Git (Yeung, 2018). Additionally, frameworks like React, Angular, or Vue.js for the front end, and Express or Django for the back end. | Programming languages include HTML, CSS, and JavaScript for front-end development. For back-end development, languages like Python, Node.js (JavaScript), or Ruby may be used (Bethencourt, 2016). Common tools and IDEs include Visual Studio Code, Atom, and Eclipse, which facilitate coding and debugging (Bethencourt, 2016). Additionally, version control systems like Git and deployment tools like Docker (Bethencourt, 2016). | Programming languages typically include JavaScript for front-end interactivity, HTML for structure, and CSS for styling (Lee, 2016). For back-end development, languages like Node.js, Python, or C# might be utilized. Commonly used (IDEs) are Visual Studio Code, WebStorm, or Visual Studio for C# (Mattwojo, 2024). Additionally, frameworks such as React or Angular for the front-end and Express.js or ASP.NET for the back end. For deployment, tools like Docker or Heroku are also valuable (Mattwojo, 2024). | To effectively develop the gaming application "Draw It or Lose It" for mobile platforms, it's essential to set up an appropriate development environment using traditional operating systems like Linux, Mac, or Windows. Ultimately, while the actual coding and testing phases will occur on one of the traditional operating systems, setting up this structured environment with appropriate languages, tools, and frameworks will ensure that "Draw It or Lose It" can be successfully developed and deployed across various mobile platforms, aligning with The Gaming Room's expansion goals (Blanco & Lucredio, 2012).  Key development tools include Visual Studio Code for its features, Sublime Text for quick edits, and WebStorm for JavaScript development. Git is crucial for version control and collaboration. Popular front-end frameworks like React, Angular, and Vue.js enhance user interfaces, while back-end frameworks such as Express for Node.js and Django for Python support API building and server-side tasks (Lee, 2024). |

## 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**: Taking into consideration the goal of expanding “Draw It or Lose It” to multiple platforms, I would recommend the windows operating platform. It has the largest user base with great support for game development tools. It offers a large community of support in distributing applications across different devices which makes it the most ideal for cross platform gaming applications (Blanco & Lucredio, 2012). by leveraging the Windows operating platform along with solid storage solutions, efficient memory management, and robust communication protocols, The Gaming Room can effectively expand "Draw It or Lose It" across multiple devices while ensuring user security and a seamless gameplay experience.
2. **Operating Systems Architectures**: The windows architecture consists of several important parts. Such as the user and kernel modes that restricts user applications ensuring stability and security. Additionally, The Windows Subsystem for Linux allows for running Linux distributions on windows add to the ease of cross platform development (Silbershcatz, 2008). Windows also proved the DirectX API which offers optimized performance for graphics in game development. Lastly windows is extremely compatible with support older applications as well as modern and can aid in the transition of existing code to newer architectures.
3. **Storage Management**: For windows a suitable storage management system would Microsoft SQL server or Azure SQL database. Both proved good database management capabilities, performance, reliability and are specifically designed for handling data being accessed simultaneously by multiple users (Chendrayan, 2020).
4. **Memory Management**: Windows has extensive memory management options that would benefit “Draw it or Lose it”. One being virtual memory that allows for applications to use more memory than what is physically available. Additionally, windows employ efficient caching mechanisms to streamline access to frequently used resources, this greatly improves the applications performance (Silbershcatz, 2008). Lastly windows use garbage collection which in managed code environments such as .NET can automatically free up memory that isn’t being utilized (Mattwojo, 2024). This reduces bugs, leaks, and overall improves the applications stability.
5. **Distributed Systems and Networks**: to facilitate effective communication between various platforms we can implement “RESTful APIs”. This would allow the game application to seamlessly interact with all clients running on different operating systems (Gupta, 2017). Additionally, web sockets can be used for real time communication to enhance gameplay performance (Yeung, 2018). The implementation of a robust message queue such as Rabbit MQ or AZURE Service Bus can help handle message dependencies, ensuring that the game is reliable during connectivity outages by storing all messages until they are processed (Chendrayan, 2020).
6. **Security**: For user protection across platforms, windows offer many capabilities such as user account control which protects against unauthorized changes to the system settings. Another capability is BitLocker, this ensures data encryption on all devices and therefore safeguards the user information from unauthorized access (Chendrayan, 2020). Implementing transport layer security for data transfer between devices Is essential to protect information in transit.

References

Angus Yeung. (2018). *Hands-On Server-Side Web Development*. Packt Publishing.

Blanco, J. Z., & Lucrédio, D. (2021). A holistic approach for cross-platform

software development. <https://doi.org/10.1016/j.jss.2021.110985>

Gregory Casamento, & Stuart Grimshaw. (2017). *Mastering MacOS Programming : Hands-on Guide to MacOS Sierra Application Development*. Packt publishing.

Gupta, L. (2023, December 12). What is REST - REST API Tutorial. Restfulapi.net. https://restfulapi.net

Lavieri, D. E. (27-04-2019). Hands-On Design Patterns with Java. [VitalSource Bookshelf 10.4.1]. Retrieved from vbk://9781789809954

Lee, N. (2024, September 11). *The Ultimate Guide to a Smooth Dev Environment Setup: Tips and Best Practices | Speedscale*. Speedscale. https://speedscale.com/blog/the-ultimate-guide-to-a-smooth-dev-environment-setup-tips-and-best-practices/

Silberschatz, A., Baer Galvin, P., & Gagne, G. (2008). *Operating System Concepts* (8th ed.) [Review of *Operating System Concepts*]. Wiley. https://learning.oreilly.com/library/view/operating-system-concepts/9780470128725/

‌ Michael Bethencourt (2016). *Game development on Linux: Using Linux at every step of the process*. (2016). Michaelb.org. <https://michaelb.org/archive/article/5.html>

‌mattwojo. (n.d.). *Setup a development environment on Windows*. Learn.microsoft.com. <https://learn.microsoft.com/en-us/windows/dev-environment/>

Chendrayan Venkatesan, & Sherif Talaat. (2015). *Windows PowerShell for .NET Developers - Second Edition : Efficiently Administer and Maintain Your Development Environment with Windows PowerShell*. Packt Publishing.