

# 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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 1/24/2025 | Grant McCord | Initial revision |
| 2.0 | 2/09/2025 | Grant McCord | Reviewed content against Week 5 rubric criteria and also reviewed feedback from first submission. Minor edits to the Evaluation section. |
| 3.0 | 2/223/2025 | Grant McCord | Reviewed previous submission feedback and updated [Recommendations] section. |

**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)

Creative Technology Solutions (CTS) is collaborating with The Gaming Room to expand their popular Android game, *Draw It or Lose It*, into a web-based application that supports multiple platforms. This project will broaden the game's accessibility and user base, meeting customer demands for cross-platform compatibility and user collaboration. The Gaming Room lacks the technical expertise to establish the required environment and will rely on CTS for delivering a streamlined development plan.

To address this need, CTS will design and develop a scalable and user-friendly web application. The solution will support multiple teams and players, enforce unique game and team names to avoid conflicts, and ensure that only one game instance runs in memory at a time. Proven software design patterns will be utilized to ensure efficiency and maintainability throughout the development process.

This solution will enhance the user experience, grow The Gaming Room’s audience, and set *Draw It or Lose It* up for long-term success in the gaming industry. The next steps include completing the software design document and writing a hardware requirements document.

## Requirements

|  |  |
| --- | --- |
| Business Requirements  *Business requirements are the high-level objectives a a solution must fulfill.* | |
| BR1 | The system shall provide a web-based version of the existing *Draw It or Lose It* game to support multi-platform access. Expanding to a web-based platform will increase accessibility for users on different devices, growing the user base and driving revenue. |
| BR2 | The web-based version of *Draw It or Lose It* shall replicate the gameplay, user interface, game rules, and experience of the original Android app to ensure a seamless transition for existing users and attract new players. Ensuring consistency with the existing app will maintain user base, minimize existing player learning curve and attract new users. |
| BR3 | The system shall ensure only one instance of the game can run at any given time. This will match customer expectations around game instance counts. |
| BR4 | The system shall enable one or more teams comprising multiple players each to participate with unique names for games, teams, and players. Multi-team functionality fosters competitive gameplay and enhances user engagement. |
| BR5 | The system shall support client needs for iterative development, with the flexibility to incorporate feedback during the process. Iterative development allows the system to adapt to client feedback and respond to customer requirement changes. |
| BR6 | The system shall be designed for scalability, allowing for future updates and growth in users. This approach will ensure long-term success and align with stakeholders’ objectives for a web based platform. |
| BR7 | Implement a user friendly interface for player and team management. This will meet stakeholder expectations for an efficient, easy-to-manage system that complements the original game. |
| BR8 | Implement a feature that checks for available game and team names, preventing conflicts and ensuring smooth gameplay setup. Validating name availability prevents conflicts during game setup, creating a better game experience. |
| BR9 | Game code shall be optimized for mobile devices to ensure fast loading times, reduced bandwidth usage, and smooth gameplay, to address the growing user base of mobile phone gamers. |

|  |  |
| --- | --- |
| Technical Requirements | |
| TR1 | The game shall be compatible across various web browsers and versions, and shall address any compatibility issues with older browsers and devices. This will require the use of various software technologies to ensure compatibility on modern and legacy browsers, as well as devices with different screen sizes and resolutions  *References:* BR1 |
| TR2 | The application shall be written in Java, utilizing appropriate frameworks and libraries to support development and performance. Compatibility testing across various Java versions and environments shall be conducted to ensure the application runs consistently on different systems and devices. |
| TR3 | The application shall allow only one instance of the game in memory at any given time, using unique identifiers for games, teams, and players. This requirement will be implemented in the system code rather than the game code. *References:* *BR3* |
| TR4 | The solution shall employ scalable technologies and frameworks to support a growing user base and future feature updates. Investigation into technologies and frameworks to support the solution will be required, as long as prototyping and testing to validate infrastructure decisions.  *References:* *BR6* |
| TR5 | The system shall employ a database, such as MySQL or MongoDB, to store player data and game state information, ensuring efficient data retrieval and updates during gameplay. |
| TR6 | The web-based game shall require secure HTTPS connections for all communication between the client and the server. This will ensure the security of user data and maintain confidentiality, all client-server interactions. Security certificates will need to be obtained and configured on the web servers. |
| TR7 | The system shall implement efficient web communication protocols for real-time communication between clients and the server, ensuring continuous synchronization throughout gameplay. |

## [Design Constraints](#_2et92p0)

|  |  |
| --- | --- |
| Design Constraints | |
| DC1 | The game shall be designed to maintain a consistent user experience across various devices, optimizing the interface for a web environment. This will involve redesigning certain aspects of the interface to accommodate different screen sizes and input methods. Early usability testing may be used to provide a feedback loop for development.  *References:* BR1, TR1 |
| DC3 | The game's state shall be saved periodically in the event of unexpected interruptions. This requires implementing real-time data persistence and state recovery. These systems must be designed to be efficient in order to prevent delays in gameplay.  *References:* TR5 |
| DC4 | The game shall be designed to handle varying internet connection speeds, ensuring smooth gameplay on slower or unstable networks. This requires additional development time to optimizing asset, implement adaptive network management, and add lag compensation in multiplayer settings.  *References:* BR1, TR1 |
| DC5 | The application shall protect user data, including game progress, player information, and team details, from unauthorized access. Developers must implement security best practices to protect data in transit and at rest, ensuring secure communication and compliance with privacy regulations.  *References:* TR6 |
| DC6 | The system should be designed to support a varying number of players and teams without performance degradation, especially as the number of users increases.  *References:* BR6 |
| DC7 | Use techniques like lazy loading or progressive loading for images and other assets to ensure that the game can start quickly, with additional assets being loaded in the background as needed. This reduces initial load time and improves responsiveness, especially for lower bandwidth connections or devices with low memory.  *References:* BR7 , BR9 |

## [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 illustrates a program designed to manage a game. It comprises key object-oriented principles such as object inheritance, the singleton design pattern, and object composition. The *ProgramDriver* class serves as the entry point with a *main()* method and uses the *SingletonTester* class to validate the singleton pattern. The *Entity* class acts as a base class for *Game*, *Team*, and *Player* classes, providing shared attributes like id and name. This inheritance structure promotes code reusability, ensures consistency across related objects, and is represented in UML as a composition relationship. The *GameService* class implements the *singleton design pattern*, ensuring only one instance of the game service exists.

The benefits of this design are significant. First, it simplifies maintenance as changes to the Entity class automatically propagate to its subclasses, reducing the need for updates in multiple locations. The system is also easily scalable, as the singleton design of the *GameService* class ensures there is a single point of management, and it can be adapted in the future to handle multiple game instances without introducing conflicts. The inheritance structure ensures consistency across the various entities (*Game, Team, Player*), which is helpful when implementing features like searching or sorting based on common attributes. Moreover, the use of inheritance and composition allows for code reusability, reducing redundancy and making the codebase more maintainable. The separation of concerns between *GameService* and the game represents encapsulation, where each is responsible for different aspects of the system, yet can interact through well-defined public interfaces, promoting modularity and making the system easier to manage and scale.

**"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** | Mac offers stability, security, and excellent integration with cloud services, making it a good option for web hosting. However, it is costlier than other platforms, and fewer hosting providers support macOS environments Whatley (2024), limiting choices. | Linux is ideal for web hosting due to its open-source nature, reliability, and cost-effectiveness. There are multiple cost effective enterprise Linux distributions available. Linux is also highly customizable for server capabilities and is very secure. | Windows supports various technologies for web hosting but is resource intensive, requiring a more robust hardware profile and also requires licenses. It is not as cost-effective as Linux. Cloud licensing and OS platform requirements drive cost up. | Mobile devices are not suitable for hosting web applications due to limited processing power and storage capacity. |
| **Client Side** | For a browser game on Mac, compatibility with macOS browsers is critical. This is a browser game and may be exempt from Mac OS strict app guidelines. | For a browser game in Linux, testing and customizing source for different versions of open source browsers (Opera, Firefox) may add costs for validation and development. | Compatibility with popular browsers on Windows such as Edge and Chrome will need to be checked. The diversity of devices and browser versions adds complexity to testing and optimization. | Mobile devices require support for various screen sizes and orientations. Code optimization will likely be required for mobile browsers for presentation and performance. |
| **Development Tools** | Xcode is the primary IDE for macOS apps, supporting Swift and Objective-C. | Linux supports a range of free, open-source tools and frameworks like Django, Flask, and Node.js. | Windows uses tools like Visual Studio and Visual Code for a range of languages such as C# and JavaScript. | iOS development relies on Xcode with Swift or Objective-C, while Android development uses Android Studio with Java or Kotlin. |

## 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**:

To host *Draw It or Lose It* as a web-based game, I recommend using a cross-platform framework like HTML5 with JavaScript. This approach ensures that the game can run on various platforms (Windows, macOS, Linux, and mobile devices) without requiring major modifications to the codebase. According to (USV 2023, September 14), "*Mobile gaming has become a major part of the entertainment industry, driven by its accessibility and convenience. Players can enjoy games anywhere and anytime, without the need for dedicated gaming consoles or PCs*." The best match to support this would be JavaScript. Per (Elitex 2023, December 6), "*In the case of mobile applications, JavaScript has become increasingly popular due to powerful tools for creating cross-platform mobile games*." The use of web technologies will enable faster development and greater compatibility across devices, allowing the game to reach users on both desktop and mobile environments through a browser. (Tech Stack, n.d.).This approach not only provides flexibility but also facilitates scaling and broadens the game’s reach to a wider audience.

1. **Operating Systems Architectures**:

The recommended architecture starts with a client-server model running on Linux for the server side. Linux provides a cost-effective, reliable, and scalable environment for hosting the game server. “*Hosting web games on Linux offers several advantages, including cost-effectiveness, stability, and scalability*.” (Wikipedia contributors, n.d.) The client-side (browser) handles the user interface and gameplay logic, while the server-side, running on Linux, manages game state, user profiles, and game interactions. Linux supports high-performance web servers like Apache or Nginx, which can efficiently handle multiple simultaneous connections. This architecture is flexible and scalable, enabling multiple clients to connect to a central server.

1. **Storage Management**

Data management will use cloud storage (such as AWS, Google Cloud, or Microsoft Azure). Cloud storage offers scalability, security, and reliability, making it ideal for handling user data, game assets, and game state. Cloud solution providers offer robust backup and disaster recovery options, ensuring that the game’s data is protected from loss. Data can be managed using NoSQL databases like MongoDB for flexible and scalable storage, particularly for user profiles and game state, which are not formatted for storage in traditional databases.

1. **Memory Management**:   
   Memory management on the client side will be handled by the browser and the JavaScript engine. Modern browsers come with built-in garbage collection, which automatically clears up memory; this helps keep things running smoothly under heavy load. To avoid hiccups during gameplay, lazy loading can be leveraged to reduce unnecessary memory use. Since this game has a large asset library, textures and models can be loaded dynamically based on what the player is doing.

On the server side, cloud hosting platforms like AWS, Google Cloud, or Microsoft Azure offer dynamic memory management. These platforms can allocate resources based on demand, allowing the game to scale as needed. During heavy load times, cloud services can spin up additional resources, and during lighter periods, they can scale down, optimizing both performance and cost.

1. **Distributed Systems and Networks**:

To enable communication between platforms, *Draw It or Lose It* can use WebSockets for real-time, bidirectional communication between the client and server. WebSockets allow the game to send and receive data instantly with minimal latency which will be crucial for a real-time multiplayer game. (Ably, n.d.). Cloud-based services provide a distributed environment where multiple instances of the game can run across different geographical locations, reducing latency and ensuring that players from different regions experience smooth gameplay. These are facilitated by data centers located across geographical regions which speed up access for users in those areas.

1. **Security**:

The game can use SSL/TLS encryption for data in transit. This will secure user interactions, such as logging in, submitting scores, and updating profiles. “*SSL/TLS protocols are essential for securing web communications by encrypting data transmitted between clients and servers, thereby protecting sensitive information from unauthorized access.*“ (DigiCert, n.d.). On the server side, secure authentication (e.g., OAuth or JWT) can be implemented to protect access and user data. “*Combining OAuth and JWT can enhance security and performance in web applications. OAuth provides strong authorization, while JWTs offer a secure and efficient way to send and validate tokens*.” (Permify, 2023). For data storage, sensitive user information should be encrypted at rest, and proper access control measures should be enforced to prevent unauthorized access to player data. Regular cloud service configuration security audits should be performed to identify and address vulnerabilities.

1. **7. Testing:**

To ensure *Draw It or Lose It* operates smoothly across all platforms and various customer devices, a comprehensive testing approach is essential. Unit testing with frameworks (Jest, Mocha) can be verify game logic, UI components, and client-side performance (Qase, n.d.). Integration tests, using tools such as Cypress or Selenium, can validate communication between the client-side and server-side, including secure authentication and data handling. Performance testing, including load tests with JMeter or LoadRunner, will assess server scalability and memory management, ensuring the game can handle traffic spikes while maintaining smooth gameplay. For security, penetration testing and regular security audits will evaluate SSL/TLS encryption, authentication protocols, and system vulnerabilities. Finally, acceptance testing (User Acceptance Tests) will gather feedback from real players to ensure the game meets user expectations and functions well across different devices and network conditions. These should be phased with early Beta level rollouts to a limited set of users, where early feedback can help shape the product. Testing can be rolled out at the earliest phases of the project and scaled as additional capabilities are deployed.

References:

UltaHost. (n.d.). *Mac VPS hosting: What it is & how to benefit from it*. UltaHost. Retrieved January 25, 2025, from <https://ultahost.com/blog/mac-vps-hosting-what-it-is-how-to-benefit-from-it/?utm_source=chatgpt.com>

Elitex. (2023, December 6). *JavaScript game development trends*. Elitex. Retrieved January 25, 2025, from <https://elitex.systems/blog/javascript-game-development-trends/?utm_source=chatgpt.com>

USV. (2023, September 14). *The rise of mobile gaming*. USV. Retrieved January 25, 2025, from <https://usv.edu/blog/the-rise-of-mobile-gaming/>

Wikipedia contributors. (n.d.). *LAMP (software bundle)*. Wikipedia. Retrieved January 25, 2025, from <https://en.wikipedia.org/wiki/LAMP_%28software_bundle%29>

Ably. (n.d.). *WebSockets: Pros and cons*. Ably. Retrieved January 25, 2025, from <https://ably.com/topic/websockets-pros-cons>

DigiCert. (n.d.). *What is SSL, TLS, and HTTPS?* Retrieved from <https://www.digicert.com/what-is-ssl-tls-and-https>

Permify. (2023, December 1). *OAuth vs JWT: Ultimate Comparison*. Permify. <https://permify.co/post/oauth-jwt-comparison/>

Whatley, S. (2024, March 27). *Comparing Mac and Windows servers for web hosting*. MamboServer. <https://www.mamboserver.com/blog/comparing-mac-and-windows-servers-for-web-hosting/>  
  
Tech Stack. (n.d.). *Web development vs. mobile app development*. Tech Stack. <https://tech-stack.com/blog/web-development-vs-mobile-app-development/>

Qase. (n.d.). *Building a robust test automation base with Mocha and Jest*. Qase. Retrieved February 23, 2025, from <https://qase.io/blog/mocha-jest-test-automation-base/>