

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/17/23 | Chantz Doan | Initial document covering Executive Summary, Design Constraints, and Domain Model. |
| 1.1 | 10/01/23 | Chantz Doan | Added information to the Evaluation section |
| 1.2 | 10/15/23 | Chantz Doan | Added information to the Recommendation 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)

The Gaming Room is looking to make a web-based version of their game, Draw It or Lose It, currently only available on Android. To do this, we need to set up a web-based environment that considers the server side, client side, and developmental tools needed to create the application while maintaining the application's functionality requirements outlined by the client.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

**User Interface Compatibility**: The game must be designed to work on different screen sizes and resolutions, from large desktop monitors to small mobile phone screens. Implementing responsive design increases the complexity of user interface development. Designers and developers will need to test the application across multiple devices, requiring additional time and resources.

**Browser Compatibility**: Different operating systems have different preferred web browsers. Cross-browser compatibility will require thorough testing and potentially extra development time to handle browser-specific quirks.

**Performance**: Different devices have different hardware capabilities. The game should be optimized to run smoothly on lower-end devices. Performance optimization may limit the complexity of game features, graphics, or functionalities that can be implemented.

**Input Methods**: Desktops may rely on a keyboard and mouse, whereas mobile devices rely on touch input. The design must accommodate these differences mechanically with event handling and visually with the user interface.

**Security:** Security measures must be robust enough to protect sensitive user data and prevent unauthorized access to game features. This may necessitate advanced encryption methods and multi-factor authentication, which can complicate the user sign-up and login processes.

**Development Tools**: Different developmental tools, such as Integrated Development Environments and programming languages, can affect development. Choosing the right development tools is crucial for long-term maintainability and can impact development speed.

## [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)

**ProgramDriver initiates the application and utilizes SingletonTester to validate the Singleton nature of the GameService class. The Entity class is a base class with general attributes inherited by Game, Team, and Player classes, illustrating the OOP principle of inheritance. GameService is designed as a Singleton responsible for game management and linking to multiple Game instances. This class also represents the principle of encapsulation by grouping game-related data and functionalities together. Each Game instance is related to multiple Teams, and each team is further linked to multiple Players, establishing zero-to-many relationships between these classes.**

**"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** | macOS provides a platform for web application hosting using macOS Server with Apache to host the server, which costs $20 for a macOS Server license plus hardware costs. MacOS's strengths lie in its seamless integration with other Apple products and suitability for iOS app backends. MacOS's weakness is that it is not ideal for large-scale, high-traffic applications. Mac servers are typically bound by the physical hardware they run on, which may include CPU power, memory, and storage limitations. This hardware-bound nature makes it difficult to quickly scale resources up or down in response to traffic spikes. macOS is Best suited for smaller projects or iOS-focused applications. | Linux provides a platform for web application hosting using Apache or Nginx to host the server. Linux's strengths are that it is highly customizable, scalable, and cost-effective, as many of the tools and utilities needed are open-source and free. Linux is the preferred choice for cloud-based applications as its lightweight nature and ability to be heavily customized make it perfect for scaling. Its weaknesses are that it may require specialized expertise in Linux system administration for optimal setup, security, and maintenance. This could be a hurdle for teams who have yet to be versed in Linux, potentially increasing the time and cost for staff training or hiring specialized personnel. Linux is most Ideal for applications that plan to scale and require customization. | Windows provides a platform for web application hosting using Windows Server with IIS. Windows's advantages lie in its integration with existing Windows infrastructure and broad support for various technologies. Window's weaknesses are that it can be costly due to licensing. These costs can quickly escalate if the game becomes popular and requires scaling. Best suited for businesses already invested in a Microsoft ecosystem. | Mobile Devices are generally unsuitable for server-side hosting due to limited resources, Limited computational power, and inability to scale up resources dynamically to meet increasing demand, making them impractical for server-side applications that need to support thousands of players. |
| **Client Side** | HTML5, CSS3, and JavaScript will be key for client-side development to ensure a modern, responsive HTML interface for the Mac desktop environment. Developers must prioritize browser compatibility by testing the application within various web browsers. This will ensure a seamless user experience across different web browsers on macOS. Special care must be taken to ensure the web APIs are flexible and robust enough to handle desktop and mobile clients for real-time gameplay and data transmission.  Regarding costs, the main expenditure will be human resources with expertise in front-end technologies, particularly those well-versed in browser compatibility issues and responsive design. The primary languages needed to expand to Mac are open-source and free, although licensing fees could still be relevant with  Third-party Libraries/Frameworks,  Testing Tools,  Fonts/Media Licenses, Design Software. | The client-side development will primarily rely on HTML5, CSS3, and JavaScript for the Linux desktop environment to ensure the application runs smoothly within various web browsers. Given Linux's fragmented nature with multiple distributions and desktop environments, specialized skills in Linux systems and browser behavior will be required. This will ensure a consistent and responsive gaming experience across Linux distributions like Ubuntu, Fedora, and Debian. This extensive testing will inevitably require time and could increase labor costs, as the team must have some experience with Linux systems and browser behavior. The primary languages are free and open source, although licensing fees could still be relevant with  Third-party Libraries/Frameworks,  Testing Tools,  Fonts/Media Licenses, Design Software. | For the Windows environment, client-side development will primarily focus on creating a modern, responsive HTML interface that can run inside various web browsers. The technologies used for this purpose would be standard web technologies like HTML5, CSS3, and JavaScript. One concern unique to the Windows environment is compatibility across different Windows versions. Windows has a broad user base, and only some are on the latest version of the OS. Ensuring the game runs smoothly across different versions will require thorough testing, which could extend the timeline and inflate costs. Older versions lack support for newer web technologies, so fallback options must be developed to handle such cases. The primary languages are free and open source, although licensing fees could still be relevant with  Third-party Libraries/Frameworks,  Testing Tools,  Fonts/Media Licenses, Design Software, Windows OS Licenses for each version of Windows you test on. | For the mobile client side, including Android and iOS, the focus should be on delivering a modern, responsive interface compatible with popular mobile web browsers like Chrome and Safari. Since the client already has an Android-based application, expanding to iOS should prioritize cross-browser and cross-platform compatibility, using HTML5, CSS3, and JavaScript as the core technologies. Enhancing user experience through touch-friendly interfaces will need to be prioritized. Overall, the primary time and cost expenditures would be on human resources skilled in responsive design, mobile UX, and cross-browser compatibility. The primary languages are free and open source, although licensing fees could still be relevant with  Third-party Libraries/Frameworks,  Testing Tools,  Fonts/Media Licenses, Design Software. |
| **Development Tools** | Mac-based development tools, HTML5, CSS3, and JavaScript, are the primary languages required to meet the client's web-based interface goals. Developers could utilize Visual Studio Code for the IDE and Git for version control. There is no need for a separate specific Mac development team, although the team will need developers to have some experience with Mac. The focus can be on hiring web developers skilled in cross-browser compatibility. The core web development languages are open source, and using Visual Studio code for our IDE, we avoid licensing costs. | Linux-based development tools, the recommended languages remain HTML5, CSS3, and JavaScript. Developers could utilize Visual Studio Code for the IDE and Git for version control. A Linux-specific development team is unnecessary, as a single team with cross-browser compatibility experience should suffice. That said, the team will still need to have experience in Linux systems for testing and optimization purposes. The core web development languages are open source, and using Visual Studio code for our IDE, we avoid licensing costs. | Windows-based development tools would continue using HTML5, CSS3, and JavaScript. The preferred IDE would be Visual Studio Code and Git for version control. Although Windows-specific expertise is optional, familiarity with browser optimizations like Internet Explorer or Microsoft Edge would be beneficial. The core web development languages are open source, and using Visual Studio code for our IDE, we avoid licensing costs. | Android and iOS platforms will use HTML5, CSS3, and JavaScript to provide a modern, responsive interface. Visual Studio Code will be the IDE, and Git will handle version control. The existing application is optimized for Android, so the development focus will shift towards iOS compatibility. There is no need for a mobile development team, as a cross-browser development team should suffice, given this is a web-based application. That said, the team will still need experience with Android and iOS for testing and optimization. The core web development languages are open source, and using Visual Studio code for our IDE, we avoid licensing costs. |

## 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**: Considering the need for scalability, customization, and the expertise required, I recommend Linux as the primary operating platform for server-side application hosting. Linux is not only open source but also offers unparalleled scalability options. Its robust community and extensive documentation mean that solutions for customization and troubleshooting are readily available. These factors make Linux a perfect fit for adapting "Draw It or Lose It" to various computing environments.
2. **Operating Systems Architectures**: Linux operates on a Monolithic Kernel architecture, which has its drivers, kernel extensions, and file systems in the same address space. This results in better performance and efficiency. Also, you can opt for various architectures like x86, ARM, and even RISC-V with Linux. This makes it versatile and suitable for various hardware environments.
3. **Storage Management**: For "Draw It or Lose It," a cloud-based storage solution like Amazon S3 is recommended due to its high availability and durability, ensuring that the pre-rendered images used in the game are reliably and quickly accessible. Integrating a Content Delivery Network like AWS CloudFront is advisable to enhance performance further and reduce latency across different geographical locations. For managing game state and metadata, an in-memory data store like Redis can be utilized for its speed and cross-platform compatibility. This setup provides a quick, reliable, and scalable storage architecture.
4. **Memory Management**: Linux uses a combination of paging and segmentation for memory management. It features demand paging, reducing the I/O required to load up processes and improving performance. With the help of Advanced Configuration and Power Interface, Linux can also put unused memory areas to sleep, conserving power. Memory management in Linux is exceptionally adaptable, catering to various applications, from lightweight to heavy-duty processes.
5. **Distributed Systems and Networks**: For Draw It or Lose It to work smoothly across different platforms, a RESTful API enables seamless data exchange between the server and the client. To facilitate real-time gameplay and instant updates, WebSockets can be used for real-time, bidirectional communication between the server and multiple clients. This allows instantaneous synchronization of game states among players. Dependencies in a distributed system include network latency, data consistency, and fault tolerance. A system of load balancers, caching, and data replication can help mitigate these dependencies. To manage potential issues like connectivity outages, a retry mechanism can be built into the RESTful API and WebSocket connections. This would allow for automatic reconnection attempts, thus enhancing the game's resilience against transient network failures.
6. **Security**: Security in a Linux environment can be fortified using various approaches, including SELinux, which can enforce mandatory access controls, thus mitigating unauthorized activities. Employing containerization techniques like Docker can isolate applications and enhance system resilience by creating a contained environment for each application component, minimizing the risks of system-wide failures, and improving maintainability. Implementing HTTPS to encrypt data sent over the network between the web server and the user's browser is crucial, providing a secure channel for users to interact with the application. This is particularly critical when transmitting sensitive user information like login credentials, creating a more trusted and secure user experience.