

<Name-of-Software-Application>

# **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 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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)

Our team is set to expand "Draw It or Lose It" from a mobile-exclusive game to a versatile web-based application that can engage players across different platforms. The core challenge is to adapt the interactive drawing game into a web environment that can efficiently support multiple teams and players simultaneously, while maintaining unique identifiers for each entity to ensure exclusivity and prevent overlaps.

The proposed solution leverages the Singleton design pattern to ensure a single instance of the game is maintained in memory, providing a consistent and controlled gaming experience. To enforce unique naming for games and teams, we will implement the Iterator pattern, facilitating seamless checks against current names. The software's structure will be guided by a detailed UML class diagram that outlines the system's components and their interactions.

Our approach prioritizes scalability and adaptability, preparing for future enhancements and varied hardware integrations. Clear and concise communication will be upheld throughout the development process, keeping the project aligned with the strategic objectives and delivering a product that meets the client's needs for a broad, inclusive gaming platform.

## Requirements

The project aims to transition "Draw It or Lose It" to a web-based platform, enhancing accessibility and player engagement across various devices. The application will be designed to adapt fluidly to different screen sizes and browser types for a consistent gaming experience on desktops, tablets, and smartphones.

Technically, the game will facilitate multiple teams and players within each session, enforcing unique identifiers for teams and game instances. It will also feature a real-time interaction system, with a focus on maintaining the integrity and pace of gameplay. A backend validation system is required to ensure the uniqueness of names chosen by users.

A single instance of the game in memory is paramount for consistency and management of game state. This will be managed using singleton and iterator patterns for efficient session control and entity navigation. These technical stipulations will frame the development and design strategy, aligning the end-product with the envisioned user experience and operational goals.

***<* 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.> I did this anyway to see if it would be close to what is expected if you do not have time to look at it I understand***

## [Design Constraints](#_2et92p0)

The primary constraint is ensuring real-time performance across diverse network conditions and devices, which requires efficient client-server communication and resource management to handle simultaneous user actions without latency.

Another significant constraint is the uniqueness of game and team names within a distributed environment where data consistency must be maintained without direct database locks, which can and will be addressed through careful design of the data validation logic and state management.

The requirement for a single instance of the game in memory at any given time imposes a constraint on the scalability model, necessitating a robust singleton pattern implementation to manage game sessions. This pattern must be designed to be thread-safe and capable of handling high traffic without bottlenecks or performance degradation.

Cross-platform compatibility is also a constraint, dictating the choice of technologies and frameworks that are universally supported and responsive. The application must be meticulously tested across all targeted platforms to ensure a uniform user experience.

These constraints directly influence the architecture, requiring a balance between a stateful application to maintain game state and stateless design principles for scalability. The chosen architecture must also support easy updates and maintenance without disrupting the live environment, ensuring the game's longevity and adaptability to future requirements.

## [System Architecture View](#_ilbxbyevv6b6)

For this idea that is presented by our team on the design of Draw it or lose it we are suggesting a three tier architecture.

The presentation tier will be built with a responsive design to ensure compatibility across various devices and browsers. It will communicate with the logic tier via API calls using AJAX for a smooth and dynamic user experience without full page reloads.

The logic tier, or the application server, will implement the game's core functionality. Here, the singleton pattern will manage the game state, ensuring a single instance in memory, while iterator patterns will navigate through collections for entity management. This tier will handle client requests, process game logic, manage user sessions, and enforce business rules such as the uniqueness of names.

The data storage tier will utilize a database system designed for high read/write speeds and concurrency control. This is crucial for maintaining the integrity of game states and user data. The database will be structured to optimize queries for game and team name checks, player statistics, and game history.

A logical topology will incorporate RESTful services for communication between the presentation and logic tiers, with JSON as the data interchange format for its lightweight nature and ease of use. Load balancers will distribute client requests evenly to ensure that no single server becomes a bottleneck.

For storage and communication, we'll use a combination of in-memory data structures to manage active games and a persistent database for long-term data storage and retrieval. This hybrid approach balances the need for performance with data durability.

While this architecture view is not a requirement for the current project, it's crucial to have a roadmap for the system's structure to guide development and future scaling efforts. This approach ensures that as we move forward with implementation, the architecture can support the application's needs and the client's business objectives.

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 provided UML class diagram illustrates the structure of "The Gaming Room" application. At its core is the Entity class, embodying the object-oriented principle of inheritance with fundamental attributes like id and name. This base class is extended by Game, Team, and Player classes, showcasing code reusability and reducing duplication.

The Game class is linked to multiple Team instances, and each Team is linked to multiple Player instances, employing aggregation to represent these one-to-many relationships. This mirrors the real-world structure of the game and its teams.

Central to the application's design is the GameService class, which is implemented as a Singleton, ensuring a single instance in memory and maintaining consistent game state. The methods within suggest the use of the Iterator pattern to manage collections of games, although this isn't explicitly depicted.

The ProgramDriver is the application's entry point, while the SingletonTester is likely used to verify the Singleton pattern's proper functioning.

In summary, the diagram leverages inheritance, aggregation, and design patterns such as Singleton and Iterator to efficiently organize the game's architecture and meet the specified software requirements.

**"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** | Development Requirements Mac Linux Windows Mobile Devices  Server Side Mac OS is known for its robust performance and security, which are critical for hosting web applications. It integrates well with other Apple products, offering a streamlined environment for development, especially for applications that may also have an iOS client component. The Unix-based architecture of Mac ensures stability and can be a favorable environment for certain technologies and programming languages. However, Mac servers can be cost-prohibitive compared to other platforms, and there might be compatibility issues with software that's primarily designed for Windows or Linux servers. It may not offer the same level of flexibility and software compatibility as Linux, especially for server applications. | Linux is flexible, secure and very highly and widley used in server environments. It is open source and this can lead to significant cost reduction while supporting a wide range of free software tools and languages. That makes it suitable for many hosting services for web based apps. It can be customized extensively to meet specific performance or security needs. And it can handle very high volumes of traffic which will be beneficial especially for a game that is or will possibly be enabled to allow players to play and share online. However, there might be a steeper learning curve for those not familiar with Unix-like environments. | Windows servers offer excellent support for .NET technologies and other Microsoft products, making them a suitable platform for hosting web applications, especially if they are built on the Microsoft stack. Windows provides a more familiar environment for many developers and supports a wide range of enterprise applications. On the downside, Windows servers can be more vulnerable to security threats if not properly managed and typically incur higher licensing costs than Linux. Windows will require more resource and energy to run effectively. | Mobile devices, while not typically used as servers, are relevant in the context of client-side considerations for a web-based application. They offer the advantages of portability and a wide user base. When considering mobile devices for web application access, it's crucial to design for a variety of screen sizes and operating systems, optimizing for touch interaction and considering mobile data limitations. However, they are limited in hosting capabilities and are not suitable for server-side processing due to their hardware constraints and mobile operating systems. Since almost everyone has and everything lives on a mobile device today, it may not be wise to develop a game platform that have mobile integrated into it. But this can be pushed off until a later date depending on the budget at this time.  Also, Ensuring a consistent user experience across various devices and operating systems requires additional development and testing efforts. Mobile devices also impose limitations on performance and bandwidth usage, which must be considered in the application design. |
| **Client Side** | For Mac, supporting multiple types of clients necessitates consideration of the development environment's compatibility with Mac OS. Development tools and software for Mac can be more expensive, potentially increasing costs. Time investment may also be significant due to the need for developers to be proficient with Mac-specific technologies and design standards, especially for applications that need to integrate seamlessly with the Mac ecosystem. Expertise in Swift or Objective-C, along with familiarity with Xcode, is crucial for Mac client development. | When it comes to Linux, developers often have access to a wide range of open-source development tools, which can reduce software costs. However, supporting multiple clients may require additional time for customization and testing across various Linux distributions. Expertise is needed in a variety of tools and languages commonly used in the Linux environment, and developers must be comfortable with less graphical and more command-line operations compared to Mac or Windows however GUI’s can be set up if needed or wanted but will take extra time, but they can be modified to the user preference whereas windows and Mac really cannot. | For Windows, the considerations for supporting multiple clients include the broad availability of development tools, some of which are specific to the Windows platform, like Visual Studio. Costs can vary depending on the licensing of Microsoft products and development tools. Time and expertise are needed to ensure compatibility across different versions of Windows, and developers often need to be skilled in .NET framework or other Microsoft-centric technologies for optimal integration. | With Mobile Devices, supporting multiple clients involves designing for different operating systems, screen sizes, and hardware capabilities. This can lead to higher costs and more time spent on development and testing to ensure a consistent user experience. Expertise in mobile development frameworks and languages, such as Kotlin for Android and Swift for iOS, is essential. Developers also need to account for the app submission and review processes of various app stores, which can affect the project timeline. The IDE may also have a learning curve for some developers. |
| **Development Tools** | For Mac, development typically involves using Xcode as the primary IDE, with Swift and Objective-C as the main programming languages for native applications. Web development can also leverage tools like Visual Studio Code, Atom, or Sublime Text. Dependency management can be handled with Homebrew, and version control is commonly managed with Git. For cross-platform development, tools such as Xamarin or Unity might be used. | On Linux, developers have a plethora of tools and languages at their disposal. Common IDEs include Eclipse, NetBeans, and JetBrains’ suite of tools like IntelliJ IDEA and PyCharm, depending on the language used, such as Java, Python, or C/C++. For web projects, technologies like PHP, MySQL, Apache, and containerization with Docker are widely used, with Git for version control and Bash scripting for automation. | For Windows, Visual Studio is the premier IDE, especially for .NET and C# development. Other popular tools include SQL Server for databases, PowerShell for scripting, and IIS for web server management. For languages, C#, VB.NET, and increasingly JavaScript with Node.js for server-side scripting are used. For cross-platform development, Microsoft's own Xamarin is often utilized, along with Unity for game development. | When it comes to Mobile Devices, Android development typically uses Android Studio with Kotlin or Java, while iOS development relies on Xcode with Swift or Objective-C. Cross-platform tools like Flutter, React Native, and Apache Cordova are essential for developing on both Android and iOS with a single codebase. For mobile web applications, responsive design is achieved with HTML5, CSS3, and JavaScript, often with frameworks like Angular, React, or Vue.js. |

## 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**: For The Gaming Room to successfully expand "Draw It or Lose It" to other computing environments, a cross-platform approach is recommended. Linux would be our recommendation as the ideal operating platform for server-side deployment due to its robustness, scalability, security, and cost-effectiveness. Linux servers are far cheaper than Mac and cheaper and more abundant than Windows. Linux supports a wide range of development tools and languages, and its open-source nature allows for extensive customization. On the client side, web technologies such as HTML5, CSS3, and JavaScript should be utilized to ensure seamless operation across all major operating systems, including Mac, Windows, and mobile operating systems like iOS and Android. This combination ensures broad accessibility while maintaining a cost-efficient and flexible development and hosting environment.
2. **Operating Systems Architectures**: The Linux platform operates on a kernel-based architecture, renowned for its stability and security, allowing for customization across various distributions while maintaining core system performance. This modularity enables the Linux server to run web applications efficiently, with the added advantage of being highly scalable and compatible with a plethora of open-source development tools. On the client side, a browser-centric architecture using HTML5, CSS3, and JavaScript ensures compatibility across all major operating systems, including Mac, Windows, and mobile OSes like iOS and Android. This approach facilitates a responsive design that adapts to different devices, ensuring a consistent user experience regardless of the device or screen size. The combination of Linux for servers and web standards for clients provides a scalable, maintainable, and platform-independent solution for "Draw It or Lose It."
3. **Storage Management**. For the recommended Linux operating platform, adopting a storage management system like MySQL or PostgreSQL would significantly benefit "Draw It or Lose It". These relational database management systems not only offer robust data storage solutions that are highly compatible with Linux but also ensure scalability and security vital for web-based applications. They stand out due to their performance capabilities, with features such as indexing, partitioning, and query optimization, which are essential for handling the large data volumes and high transaction rates typical in gaming environments. Both MySQL and PostgreSQL support data integrity and reliability through mechanisms like transaction logs, rollback, commit features, and replication capabilities, ensuring accurate recording of player data and game states. Additionally, they offer stringent security measures, including encryption options, access controls, and protections against SQL injection, safeguarding user data and maintaining game integrity. Their widespread use and support enhance reliability and ease integration, making them ideal choices for the database needs of "Draw It or Lose It", facilitating seamless scalability and robust data management as the game grows.
4. **Memory Management**: The recommended Linux operating system employs advanced memory management techniques that are well-suited for managing the "Draw It or Lose It" software. Linux uses a combination of virtual memory, demand paging, and swap space to efficiently utilize the system's RAM. Virtual memory allows the system to use disk storage as an extension of RAM, enabling large applications to run on hardware with limited memory resources. Demand paging ensures that only the necessary parts of an application are loaded into memory, which optimizes the use of available RAM and improves application performance. Additionally, Linux's memory management system includes features like cache and buffer management, which help to reduce I/O operations, leading to faster response times for a web-based application such as "Draw It or Lose It"
5. **Distributed Systems and Networks**: To enable "Draw It or Lose It" to communicate across various platforms, a distributed software approach using a client-server model is key. In this setup, the server hosts the game logic and data, while clients on diverse platforms — like PCs, tablets, and smartphones — interact with the server via the internet. RESTful APIs on the server side facilitate this cross-platform communication, sending and receiving data in a universal format like JSON, ensuring compatibility regardless of the client device. However, this system's effectiveness hinges on reliable network connectivity. Users' experience may vary based on their internet connection quality, with potential issues like lag or disconnection. To address this, the game could be designed to handle temporary connectivity losses, synchronizing data when the connection is stable again. On the server side, implementing load balancers and redundant servers can manage traffic loads and provide contingency during outages, ensuring consistent availability and a smooth gaming experience for all users.
6. Security:

Security is paramount in protecting user information for "Draw It or Lose It," especially when operating across various platforms. Using the Linux operating system as the server-side platform enhances this aspect, as Linux is known for its strong security features. To safeguard user data, several measures should be implemented.

First, all data transmission between the client devices and the server should be encrypted using protocols like TLS (Transport Layer Security). This ensures that any data sent over the internet, such as user credentials or gameplay data, is secure and unreadable in transit. For user authentication, implementing OAuth or similar secure authentication protocols provides a robust mechanism that doesn't require storing sensitive password data on the server.

On the server side, Linux's inherent security features, such as SELinux (Security-Enhanced Linux), can be leveraged for additional protection. Regular updates and patches should be applied to the server to protect against known vulnerabilities. Moreover, firewalls and intrusion detection systems should be in place to monitor and prevent unauthorized access.

In terms of data storage, encrypting sensitive user data stored in the database protects it even if there is a breach. Regular backups and a well-planned disaster recovery strategy will further ensure data integrity and availability.

Finally, considering client-side security, especially on mobile devices, is crucial. This includes securing the app against common threats like data leakage, improper session handling, and ensuring that the app only requests necessary permissions to function.

By combining these security measures with the inherent capabilities of the Linux platform, "Draw It or Lose It" can offer a secure environment for its users, protecting their information both in storage and in transit across different platforms.