

<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 | Oct 16 2023 | Jacob Batrano | Updates to the recommendations have been made. |

**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 venturing into the realm of interactive online games with the intention to bring to life "Draw It or Lose It", an engaging game inspired by the 1980s television game "Win, Lose or Draw". The challenge, however, revolves around translating this classic concept into a web-based format that can accommodate multiple teams, each having multiple players, all competing in real-time.

Our solution offers a web-based platform tailored for "Draw It or Lose It". This design incorporates a flexible structure that can manage multiple teams and players simultaneously. A key feature ensures that both game and team names remain unique, eliminating potential confusion and enhancing the user experience. With the implementation of the Singleton design pattern, we guarantee efficient memory utilization by ensuring that only one instance of the game remains active in memory at any given time. This design caters to a broad audience by being responsive, meaning players from diverse devices - be it desktops or mobile phones - can participate without any compromise on their gaming experience.

For the client, the paramount takeaway is that this design not only embodies the core essence of the game but also incorporates modern software design principles to ensure a seamless and engaging user experience. Before embarking on the developmental phase, it's crucial to understand that the game's real-time nature requires robust backend support, and the unique name constraint might involve slight overheads. Nonetheless, these aspects are integral for the game's success and should be viewed as foundational investments.

## Requirements

**Business Requirements:**

1. **Interactive Experience:** The game "Draw It or Lose It" should provide an immersive and interactive experience reminiscent of the 1980s television game "Win, Lose or Draw".
2. **Multiple Teams and Players:** The game should support the creation of one or more teams. Each team should be capable of having multiple players assigned to it.
3. **Unique Naming:** Both game and team names must be distinct to prevent confusion and enhance user experience. The system should offer a mechanism to check the availability of a name during the team creation phase.
4. **Time-Constrained Rounds:** A game comprises four rounds, each lasting one minute. The drawings should be fully complete by the 30-second mark. If a team fails to guess correctly within the allocated time, other teams should be granted a chance to guess within a 15-second window.
5. **Large Library of Stock Drawings:** To offer a variety of puzzles and to ensure longevity and replayability, the game should pull images from a large library of stock drawings.
6. **Real-Time Competition:** After one team's time expires, remaining teams must have the opportunity to provide their guesses in real-time.

**Technical Requirements:**

1. **Singleton Design:** To manage resources efficiently and ensure consistent gameplay, only one instance of the game should be allowed in memory at any given time.
2. **Unique Identifiers:** Each instance of a game, team, or player should have a unique identifier to prevent any potential confusion and ensure data integrity.
3. **Web-Based Platform:** The game should be designed for a distributed web-based environment, ensuring accessibility across a variety of devices, including desktops, tablets, and mobile phones.
4. **Robust Backend Support:** Given the real-time nature of the game, the backend should be robust enough to handle simultaneous guesses, time tracking for rounds, and rendering images from the stock library without lags or hitches.
5. **Responsive Design:** The game interface should be adaptive to ensure players on any device experience optimal graphics and gameplay.
6. **Secure Data Management:** Player data, team configurations, and game statistics should be securely stored and easily retrievable, ensuring data protection and compliance.

## [Design Constraints](#_2et92p0)

1. **Browser Compatibility:** The game, being web-based, needs to be compatible across various browsers like Chrome, Firefox, Safari, Edge, etc. This means employing standardized web technologies and practices and frequent testing across these platforms.
   * *Implication:* Development might require polyfills or fallbacks to ensure functionality across older browser versions, possibly increasing the development time and complexity.
2. **Latency and Real-time Interaction:** A distributed environment can introduce latency. The real-time nature of guessing and round timing could be affected by network delays.
   * *Implication:* Developers will need to employ efficient algorithms, caching mechanisms, and possibly edge computing solutions to minimize latency.
3. **Database Consistency:** With multiple players interacting simultaneously, ensuring data consistency (like updating scores, team guesses) is paramount.
   * *Implication:* A robust database system with transaction handling and concurrency controls will be essential.
4. **Scalability:** As the player base grows, the system must be able to handle more concurrent users without performance degradation.
   * *Implication:* This necessitates a scalable architecture, possibly employing microservices, containerization, and cloud-based scaling solutions.
5. **Data Security and Privacy:** Player data, team configurations, and game scores need to be securely stored, complying with data protection regulations.
   * *Implication:* Robust encryption methods for data at rest and in transit, as well as adherence to standards like GDPR, will be crucial.
6. **Responsive Design Constraints:** Ensuring that the game is playable on devices of all sizes—from desktop monitors to mobile phones—requires a responsive design.
   * *Implication:* This may limit certain design choices due to screen real estate constraints, especially on mobile devices.
7. **Singleton Pattern Complexity:** Ensuring only one instance of the game in memory can introduce technical challenges, especially in a distributed web environment where multiple server instances might be running.
   * *Implication:* Careful implementation of the Singleton pattern and potential use of server orchestration tools will be needed to maintain this constraint.
8. **Asset Management:** The game relies on a large library of stock drawings. Efficiently managing, retrieving, and rendering these assets in real-time is a constraint.
   * *Implication:* A content delivery network (CDN) or similar solutions will be required to speed up asset delivery.
9. **Interactive UI/UX:** Given the interactive nature of the game, the UI/UX should be intuitive and user-friendly.
   * *Implication:* Significant time and resources will be invested in UI/UX design and testing to ensure a smooth player experience.
10. **Maintenance and Extensibility:** As with all web applications, regular maintenance to address potential bugs, updates, and new feature implementations will be needed.
    * *Implication:* Adopting a modular architecture will be critical to ensure ease of updates and potential feature extensions in the future.

## [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 provides a blueprint for the game, "Draw It or Lose It". Here's an explanation of each class:

1. **ProgramDriver Class**:
   * Serves as the starting point for the application with its **main()** method.
2. **SingletonTester Class**:
   * Validates the implementation of the Singleton design pattern. The **ProgramDriver** uses this class, which is indicated by the **<<uses>>** relationship.
3. **Entity Class**:
   * A base class that represents generic items in the system, holding common attributes like **id** and **name**. Classes such as **Game**, **Team**, and **Player** inherit properties from this class.
4. **GameService Class**:
   * Orchestrates the creation and management of games, players, and teams. Designed as a Singleton to ensure only one instance exists.
5. **Game Class**:
   * Represents individual games and holds a list of participating teams. It allows the addition of teams and can also describe itself as a string.
6. **Team Class**:
   * Represents individual teams. Maintains a list of its players. Can add players and represent itself as a string.
7. **Player Class**:
   * Represents an individual player and can express its details as a string.

**Inter-relationships**: - **GameService** has a one-to-many relationship with **Game**, depicted as "0...\*". A single **GameService** can manage multiple games. - Similarly, **Game** has a one-to-many relationship with **Team**, and **Team** with **Player**.

**Object-Oriented Principles in the Diagram**: - **Encapsulation**: Classes have private data attributes and expose only necessary methods. - **Inheritance**: Classes like **Game**, **Team**, and **Player** inherit from the **Entity** class, promoting code reuse. - **Polymorphism**: Derived classes can modify inherited methods for specific needs. - **Singleton Pattern**: Employed by **GameService** to maintain a singular instance. - **Aggregation**: Seen in the relationships between classes, indicating a composite structure.

**"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:** While MacOS does offer server capabilities through macOS Server, it might not be the primary choice for large-scale web hosting solutions. macOS Server provides integrated tools for device and application management, particularly within Apple's ecosystem. However, its scalability and performance in the context of hosting a web application for a massive audience like our game might require additional investigation. As for licensing costs, given Apple's premium branding, they may lean towards the higher side, but a detailed evaluation would be necessary. | Linux is a well-regarded choice in the server environment, especially for web-based applications like "Draw It or Lose It". The open-source nature of Linux provides flexibility and adaptability. The stability and security of Linux servers are commendable. Since Linux is open-source, there are many free distributions available. However, if opting for commercial distributions like Red Hat Enterprise Linux, licensing can range from $300 to $1,200 per year, depending on the level of support required. | Microsoft's solution, offers a robust environment for hosting applications. With its user-friendly interface and integration with Microsoft technologies, "Draw It or Lose It" could find a dependable hosting foundation here. Licensing costs for Windows Server can vary, but on average, prices might range from $500 to $6,200, depending on the edition and licensing model chosen. | While mobile devices are generally consumer-facing, there are ways to use them for specific server tasks. However, for a game like "Draw It or Lose It" anticipating thousands of players, leveraging mobile devices as servers might not be ideal. The strengths of mobile devices are in their portability and user experience. While direct hosting might be negligible in cost, the supporting infrastructure for such a setup might prove to be more challenging and less cost-effective. |
| **Client Side** | For "Draw It or Lose It" to run smoothly on Mac systems, there's a need to ensure compatibility with Safari, the default web browser, though Chrome and Firefox are also popular on Mac. Development considerations include:  **Cost:** Licensing for development tools, if any specific Mac tools are needed. Typically, web development tools like Visual Studio Code or Atom are free.  **Time:** Ensuring the game's user interface is intuitive and adheres to Mac's design philosophies may require additional design and testing time.  **Expertise:** Familiarity with Mac's behavior and nuances is crucial. Developers must be adept at cross-browser testing, especially on Safari. | On Linux, the web environment is diverse, with several browsers like Firefox, Chrome, and even niche ones like Midori. Development considerations include:  **Cost:** Most development tools for Linux are open-source and free, which could lead to potential cost savings.  **Time:** Due to the diverse range of browsers and Linux distributions, thorough testing might be time-consuming.  **Expertise:** Linux generally requires a steeper learning curve. Developers would benefit from experience in various Linux distributions and browsers to ensure "Draw It or Lose It" performs consistently. | With a significant market share, Windows is critical. Edge and Chrome are the predominant browsers. Development considerations include:  **Cost:** Licensing for certain Microsoft-specific development tools might be needed, though many web development tools are free.  **Time:** Ensuring the game interface aligns well with Windows' design and behaves consistently across its browsers might add to development time.  **Expertise:** Developers should be familiar with Windows' ecosystem and be proficient in cross-browser testing on Windows platforms. | Supporting both iOS and Android is essential. The game will run within browsers but needs to feel native. Development considerations include:  **Cost:** There might be fees associated with developer accounts for app stores if a dedicated app is considered in the future. Tools like React Native or Flutter might be used for a more native feel, which could also have associated costs.  **Time:** Mobile devices vary widely in display sizes and capabilities. Extensive testing is required to ensure "Draw It or Lose It" offers a consistent experience.  **Expertise:** Developers should be well-versed in responsive design principles and potentially familiar with mobile development frameworks. Knowledge of both iOS's Safari and Android's Chrome browser behaviors is crucial. |
| **Development Tools** | For deploying web applications on Mac, the dominant languages remain JavaScript, HTML, and CSS for frontend. For backend, Python, Ruby, and Node.js are popular choices.  **Tools:** Xcode for any Mac-specific development, Visual Studio Code, Atom, and Sublime Text for general development.  **Impact:** If "Draw It or Lose It" requires any Mac-specific features, developers familiar with macOS and its nuances would be beneficial.  **Licensing Costs:** Most web development tools for Mac are free. Xcode, Apple's primary development tool, is also free, but there's an annual fee if you wish to publish apps to the App Store. | **Linux:** Web development on Linux is similar to other platforms with languages like JavaScript, HTML, CSS, and backend technologies like Python, PHP, or Ruby.  **Tools:** Eclipse, Visual Studio Code, JetBrains suite (like PhpStorm, PyCharm), and Vim/Emacs for the enthusiasts.  **Impact:** Given the open-source nature of Linux, developers often need to be adept at handling different distributions and environments.  **Licensing Costs:** Many tools on Linux are open-source and free. However, some JetBrains tools have licensing fees. | Frontend development remains consistent with JavaScript, HTML, and CSS. Backend might lean towards .NET technologies, though Python, Node.js, and others are widely used.  **Tools:** Visual Studio is predominant, but Visual Studio Code, JetBrains suite, and Notepad++ are also popular.  **Impact:** Integration with other Microsoft services (like Azure) might require specific expertise.  **Licensing Costs:** Visual Studio has both free and professional (paid) editions. JetBrains suite and some other tools might also have licensing costs. | For a web-based game like "Draw It or Lose It", standard web technologies (JavaScript, HTML, CSS) suffice. However, if native or hybrid development is considered:  **Languages:** Swift for iOS, Kotlin or Java for Android. Hybrid solutions might use JavaScript.  **Tools:** Android Studio for Android development, Xcode for iOS, and frameworks like React Native or Flutter for hybrid solutions.  **Impact:** Mobile development requires knowledge of mobile UI/UX, device capabilities, and potentially multiple languages.  **Licensing Costs:** Android Studio and Xcode are free, but there's a one-time $25 fee for Google Play Store and an annual $99 fee for Apple's Developer Program. Some hybrid frameworks might have additional 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 Systems Architectures**:

After researching and comparing the potential platforms, I've concluded that **Linux** stands out as the most suitable server operating platform for expanding "Draw It or Lose It" across diverse computing environments. Here's my thought process:

**Versatility and Compatibility:** Through my studies, I've come to appreciate how adaptable Linux is. It doesn't matter if we're looking at PCs, mobile gadgets, or even future-forward embedded systems; Linux offers a wide range of distributions tailored for these scenarios.

**Cost Concerns:** From a budget standpoint, Linux has a significant edge. It's open source, which means we won't be bogged down by licensing fees. This seems especially advantageous for a growing initiative like "Draw It or Lose It."

**Thinking of Scalability:** As our game gains traction and more players join, our hosting infrastructure must keep up. From what I've learned, Linux has been the backbone of many large-scale web applications due to its scalability.

**Community and Support:** One thing that caught my attention about Linux is its vibrant community. Having such a vast and active community could be a real asset, offering us timely updates and solutions to any challenges we might face.

**Security Matters:** From my understanding, Linux tends to be on the safer side when compared to some other options. Its permission-based structure and the controlled environment could offer us a more secure platform for our web application.

**Integration Possibilities:** Given the array of tools and flexibility Linux offers, from integrating with databases to using container tools like Docker, it seems like it provides everything we'd need for our game's expansion.

Taking all these points into consideration, Linux feels like a robust choice that can effectively support the expansion and evolution of "Draw It or Lose It" across various computing landscapes.

**Storage Management**:

When considering the Linux operating platform and the requirements of "Draw It or Lose It", it's imperative to choose a storage management system that's both scalable and reliable. Given my exploration and understanding so far, I'd recommend using **PostgreSQL** as our primary storage management system. Here's my rationale:

**Reliability and Performance:** PostgreSQL is known for its high reliability and performance. It has a proven track record of handling large volumes of data without compromising on speed, which is essential as our game grows.

**Open Source:** Similar to the Linux operating system, PostgreSQL is open source. This means we can benefit from continuous community-driven improvements and avoid exorbitant licensing fees.

**Extensibility:** What I find appealing about PostgreSQL is its extensibility. It supports both structured and unstructured data, allowing us to adapt to the game's evolving data requirements. Additionally, with its support for JSON and JSONB, we can store game-related data in a flexible manner.

**Compatibility with Linux:** PostgreSQL has excellent support and compatibility with Linux, ensuring smooth operations without frequent compatibility issues.

**Advanced Features:** From my studies, I've learned that PostgreSQL comes with a suite of advanced features, including table partitioning, point in time recovery, and sophisticated locking mechanisms. These will be invaluable for ensuring the game's data is managed efficiently.

**Scalability:** As "Draw It or Lose It" scales with more players and more game data, PostgreSQL offers mechanisms to scale both vertically and horizontally. This flexibility ensures we can manage growing data demands without hiccups.

**Backup and Recovery:** Data safety is paramount, and PostgreSQL offers robust backup and recovery options. This ensures that our game data remains safe and can be recovered swiftly in unforeseen scenarios.

**Memory Management**:

For "Draw It or Lose It", ensuring smooth gameplay and responsiveness is crucial. A large component of this is how our chosen operating platform, Linux, manages memory. While I'm still building my expertise in this area, here's what I've gleaned regarding Linux's memory management techniques, and how they would benefit our game:

**Virtual Memory System:** One of the hallmarks of Linux's memory management is its use of virtual memory. This ensures that each process in the system believes it has its private memory, even though multiple processes might be sharing the same physical memory. For "Draw It or Lose It", this means that even if multiple instances or modules of the game are running, each would operate smoothly without directly interfering with others.

**Page Replacement Algorithms:** Linux employs advanced page replacement techniques like the Least Recently Used (LRU) algorithm. This ensures that memory pages that aren't frequently used can be replaced efficiently. This adaptability ensures our game remains responsive, especially during peak times.

**Swapping:** If physical memory starts running low, Linux uses a method called swapping to free up space. It shifts inactive processes from RAM to a dedicated swap space on the hard drive. This mechanism helps in ensuring that active processes for "Draw It or Lose It", like real-time gameplay or leaderboard updates, receive priority and maintain responsiveness.

**Demand Paging:** Rather than loading an entire program into memory, Linux employs demand paging where only the necessary pages are loaded. This minimizes memory usage and ensures that our game utilizes resources only when needed, optimizing performance.

**Memory Caching:** Linux intelligently caches frequently accessed data in memory. For a game like "Draw It or Lose It", this means that recurrent assets or modules are loaded faster, offering a seamless experience to players.

**Overcommit Handling:** Linux has a memory overcommit mechanism that helps in situations where allocated memory might exceed available physical memory and swap space. This is especially useful for ensuring that "Draw It or Lose It" doesn't crash during high-load scenarios.

**Slab Allocator:** For frequently used objects of the same size, Linux uses the slab allocator. This reduces fragmentation and ensures that memory allocations and deallocations for our game are efficient.

1. **Distributed Systems and Networks**:

Distributed systems connect multiple computers together, enabling them to work cohesively and share resources. For "Draw It or Lose It", this means that players on different platforms, whether desktop or mobile, can interact seamlessly. To achieve this, a centralized server or a cluster of servers would be crucial. These servers would manage game logic, player data, and other essential elements, communicating with clients across platforms using standard protocols such as HTTP or WebSockets.

Challenges arise in distributed systems, especially when considering the dependencies between components. If one server fails, it has the potential to disrupt the entire gaming experience. Thus, implementing redundancy, failover strategies, and load balancing are essential. Network connectivity must also be prioritized to ensure low latency for real-time interactions. Using content delivery networks (CDNs) or positioning servers in strategic geographic locations can reduce latency. Addressing potential connectivity issues and ensuring a robust system are vital for the seamless operation of "Draw It or Lose It" on multiple platforms.

1. **Security**: **Security**:

Ensuring the security of user information is paramount, especially in a gaming environment where personal details and, in some cases, financial transactions may be involved. For "Draw It or Lose It", deployed on a Linux server platform, there are several security measures we can employ:

**Encryption**: All data transferred between the client and server, be it gameplay data or personal information, should be encrypted using protocols like TLS (Transport Layer Security). This ensures that even if data is intercepted during transmission, it remains unintelligible to unauthorized parties.

**Secure Storage**: On the server side, any sensitive user information, such as passwords, should be stored in a hashed and salted form, using robust cryptographic algorithms. This ensures that even in the event of a data breach, the stored data cannot be easily deciphered.

**Firewalls and Intrusion Detection Systems (IDS)**: Implementing a firewall can filter out malicious traffic, while an IDS monitors the network for any suspicious activities, allowing us to respond to potential threats in real time.

**Regular Updates**: Keeping the server and all software components up to date is crucial. Linux distributions regularly release patches for known vulnerabilities, and ensuring timely updates can shield the system from many known threats.

**Role-based Access Control (RBAC)**: Limit access to server resources based on roles within the organization. This means that only individuals with the necessary permissions can access specific parts of the system, reducing the risk of internal threats.

**Two-factor Authentication (2FA)**: For users accessing any administrative aspects of the game or their profiles, implementing 2FA can add an additional layer of security, ensuring that even if a password is compromised, unauthorized access is still prevented.