

Draw It or Lose It

# **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 | 05/23/2025 | Rita (Nathaniel) Tetterton | The first iteration of the application. |

**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) has embarked on a new project with The Gaming Room, a client seeking to expand their existing mobile game, "Draw It or Lose It," into a web-based platform accessible across multiple devices. This game, inspired by the classic television show "Win, Lose or Draw," involves teams competing to guess drawings rendered from a vast library of stock images. Each game consists of four rounds, with a time limit of one minute per round, fostering a dynamic and engaging gameplay experience.

The Gaming Room's team requires assistance in establishing the necessary development environment to facilitate the transition from a mobile app to a web application. As the Technology Consultant, your role involves preparing a comprehensive software design document and initiating the development of the game application, ensuring that all software requirements are meticulously addressed.

## Requirements

The Gaming Room has outlined several key business and technical requirements for the development of the web-based version of “Draw It or Lose It.” These requirements are essential for ensuring that the application meets the users expectations and operates effectively across multiple platforms.

* Multi-Platform Accessibility: The game must be accessible on various platforms, including desktops and mobile devices, to reach a wider range of audiences.
* Engaging User Experience: The application should provide an engaging and interactive experience that retains the core gameplay mechanics of the original mobile app.
* Brand Consistency: The web version must maintain the branding and visual identity of The Gaming Room to ensure a cohesive user experience.
* Multi-User Support: The application must support multiple teams, each with several players, allowing for competitive gameplay.
* Unique Game and Team Names: The system must enforce unique names for games and teams to prevent conflicts and enhance user experience.
* Single Instance Management: Only one instance of the game should exist in memory at any time, which can be achieved through unique identifiers for each game, team, and player.
* Timer Functionality: Implement a timer for each round and guessing phase to ensure structured gameplay.
* Scalability: The architecture should allow for future enhancements, such as additional game modes or features, to adapt to user needs.
* Data Management: Efficient data management practices must be in place to handle user sessions, game states, and interactions seamlessly.

By addressing both the business and technical requirements, CTS can ensure that the web-based version of “Draw It or Lose It” not only meets the expectations of The Gaming Room but also provides a robust and enjoyable gaming experience for users.

## [Design Constraints](#_2et92p0)

When developing the web-based version of “Draw It or Lose It” several design constraints must be considered. These constraints will influence the architecture, functionality, and overall user experience of the application.

* Network Latency and Bandwidth: The application must be optimized for varying network conditions, as users may experience different levels of latency and bandwidth. This requires efficient data transmission and minimal server requests to ensure smooth gameplay.
* Cross-Platform Compatibility: The game must function seamlessly across different devices and browsers. This necessitates the use of responsive design principles and thorough testing on multiple platforms to ensure consistent performance and user experience.
* Scalability: The architecture should be designed to handle an increasing number of users and game instances without degradation in performance. This may involve using cloud services and load balancing techniques to manage user traffic effectively.
* Security: Providing user data and game integrity must be protected against all unauthorized access and attacks. Implementing secure authentication methods, data encryption, and regular security audits will be essential to safeguard user information.
* Real-Time Interaction: The game requires real-time interaction among players, which necessitates the use of WebSocket or similar technologies to facilitate instant communication. This adds complexity to the development process, as it requires careful handling of concurrent user actions.
* Resource Management: Efficient management of server resources is crucial to ensure that only one instance of the game runs in memory at any time. This requires a robust system for tracking game states and user sessions, which can complicate the application architecture.
* User Experience Design: The design must prioritize user engagement and ease of use. This involves creating intuitive interfaces and ensuring that gameplay mechanics are easy to understand, which may require iterative design and user testing.
* Compliance with Standards: The application must comply with relevant industry standards and regulations, such as data protection laws (e.g., GDPR). This may require additional development efforts to ensure that user data is handled appropriately.

## [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 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.

The UML class diagram for "The Gaming Room" system outlines the various classes and their interrelationships within the application. At the core of the design is the **GameService** class, which acts as the central management entity for the gaming system. This class is responsible for overseeing multiple **Game** instances, each representing a unique game session.

The **Game** class aggregates multiple **Team** objects, allowing for a structured competitive environment. Each **Team** can consist of several **Player** objects, facilitating dynamic interactions and gameplay. This hierarchical structure illustrates how the gaming system is organized, with the **GameService** serving as the parent entity that manages games, and each game further managing teams and players.

The **Entity** class serves as a base class, providing common attributes such as **id** and **name** that are inherited by the **Game**, **Team**, and **Player** classes. This demonstrates the principle of inheritance, allowing for code reusability and a cleaner design.

In terms of relationships, the diagram highlights composition and aggregation. The **GameService** has a composition relationship with the **Game** class, indicating that a game service is composed of multiple games. Similarly, the **Game** class has a composition relationship with the **Team** class, and the **Team** class has a composition relationship with the **Player** class. This structure ensures that the lifecycle of teams and players is tied to the game they belong to, reinforcing the integrity of the game state.

The **GameController** class plays a crucial role in orchestrating the game flow, handling actions such as starting rounds and processing player interactions. It interacts with various services, including the **ScoreService**, which manages the scoring logic, and the **TimerService**, responsible for tracking the duration of each round. The **DrawingService** is also integral, as it selects and renders drawing clues from a repository of stock images, enhancing the gameplay experience.

This design exemplifies several object-oriented programming principles:

* **Inheritance** is demonstrated through the relationships between classes, allowing subclasses to inherit common attributes and behaviors from their parent classes.
* **Encapsulation** is evident as classes manage their internal states and expose only necessary methods for interaction, promoting data privacy.
* **Abstraction** is utilized to simplify complex functionalities, allowing users to interact with high-level operations without needing to understand the underlying implementation details.

Overall, this UML class diagram effectively captures the architecture of "The Gaming Room" system, showcasing how various components work together to fulfill the software requirements while adhering to key object-oriented principles.

## [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** | This Unix-based system is secure and stable, providing a strong foundation for hosting various web applications. It is developer-friendly, featuring an easy-to-follow structure. However, the hardware costs are higher, and it offers limited scalability compared to other systems such as Linux and Windows. | Linux is an open-source operating system that offers greater flexibility and customization options. However, it does have some limitations with its graphical user interface (GUI), particularly regarding hardware compatibility. Despite this, Linux provides a wide range of software and is known for its scalability and reliability, all while maintaining high security. | Windows operating system boasts a high degree of compatibility with a vast array of software applications and offers extensive support for a wide range of hardware configurations. Additionally, it benefits from a robust ecosystem for developers, enabling the creation and deployment of diverse applications. However, it is important to note that Windows has been associated with significant security vulnerabilities, which can pose challenges to system integrity and user safety. | Mobile devices provide unparalleled mobility and convenient access to information and services. Equipped with touchscreen and gesture-based interfaces, these devices facilitate intuitive user interactions. Additionally, they boast a diverse array of hardware capabilities, including advanced processing power, high-resolution cameras, and various sensors. However, it is important to note that the compact size of the screen can impose certain limitations on usability and the overall experience, particularly when it comes to tasks that require extensive visual or textual engagement. Balancing these strengths and constraints is essential for optimizing the functionality of mobile devices in various applications. |
| **Client Side** | The Mac platform offers an intuitive and visually appealing interface, which enhances user experience and reduces the learning curve for new users. However, developing and maintaining multiple client versions can increase costs and complexity, requiring diverse expertise to manage different environments. | Linux provides a free and open-source environment, which is beneficial for developers. However, it may present a steeper learning curve for users unfamiliar with its interface. The need for diverse expertise can also arise when developing for various Linux distributions, which can complicate the client-side development process. | Windows is known for its wide range of software compatibility and a strong developer ecosystem. However, licensing costs can be higher compared to open-source alternatives, and the need for different versions can complicate client-side development. | Mobile platforms focus on touch-based interactions, providing a unique user experience. However, developers must consider responsive design and connectivity limitations, as well as the diverse hardware capabilities of different devices. Native features like GPS and camera access can enhance the application but also add complexity to development. |
| **Development Tools** | Development on Mac typically utilizes Node.js and JavaScript, with popular IDEs such as Visual Studio Code and Xcode. This environment supports a rich ecosystem for web application development, making it easier for developers to create and maintain applications. | Linux is favored for its robust command-line interface and package management systems like apt or yum. Developers often use a variety of IDEs, including Visual Studio Code, Atom, and Sublime Text, which cater to different programming languages and preferences. | Windows development often involves C# and the .NET framework, which are popular for building web applications. Visual Studio and JetBrains IDEs are commonly used, providing powerful tools for debugging and application management. | For mobile development, languages like Kotlin, Swift, and Objective-C are commonly used alongside Java and JavaScript. Android Studio and Xcode are the primary IDEs, offering emulators and simulators to test applications across various devices. |

## 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's expansion of "Draw It or Lose It" to other computing environments, I recommend using Linux as the operating platform. Linux is well-known for its open-source nature, which not only reduces costs but also fosters a community-driven development model that continuously enhances its features. The platform's high customizability means developers can tailor the operating environment to suit specific game requirements, allowing for optimizations that can improve performance. Linux consistently receives updates that add support for new technologies and enhance existing features. This includes support for a broad range of applications—from frontend frameworks to backend services—making it especially suitable for a cross-platform game that needs to function seamlessly on various devices, including web and mobile. The compatibility with various tools and libraries, alongside a plethora of community support, adds to Linux's appeal as the ideal choice for a game development environment.
2. **Operating Systems Architectures**: Linux operates on a monolithic kernel architecture, integrating key functionalities like process management, memory management, and device drivers into a single kernel. This design allows for high performance and efficiency, with system calls executed rapidly since they do not require switching between user mode and kernel mode, which can slow down the system. Additionally, the modularity of the Linux kernel allows The Gaming Room to include or exclude features as needed, optimizing resource consumption. Supporting various distributions such as Ubuntu, CentOS, and Fedora means developers can choose a version that is best suited to their development and deployment requirements, tailoring it based on performance benchmarks, community support, and familiarity. Moreover, the modular nature of Linux distributions allows for easy updates and patches, which can be crucial in a fast-evolving industry like gaming.
3. **Storage Management**: For storage management, I recommend PostgreSQL as the database management system. PostgreSQL is a potent, open-source relational database known for its robustness and reliability, making it an excellent choice for complex applications. It offers advanced features such as ACID compliance, which ensures reliable transactions, and support for complex queries, allowing developers to retrieve and manipulate data efficiently. Particularly for "Draw It or Lose It," PostgreSQL can effectively handle diverse types of data, including user profiles, game states, and scores, all of which require steady, consistent access and manipulation. Its extensive support for JSON data types also allows for flexible schema designs that can evolve with the needs of the game. PostgreSQL’s scalability ensures that as the user base grows and more data is generated, the database can be optimized and scaled up horizontally or vertically to accommodate increased load without compromising performance.
4. **Memory Management**: Linux’s memory management techniques significantly benefit the "Draw It or Lose It" software. The virtual memory system allows applications to utilize more memory than what is physically available by swapping segments of data to and from disk storage. This is essential during peak usage when multiple users engage with the game simultaneously. The implementation of paging—where memory is divided into fixed-size units makes it easy to manage memory efficiently. Segmentation further enhances this by allowing different programs to be loaded into separate segments, thus isolating them from each other to prevent interference. This isolation is critical for game stability, reducing the risk of crashes or performance issues during intense gameplay. Additionally, the system’s ability to dynamically allocate memory ensures that resources are used effectively, enabling smooth operation even under heavy load.
5. **Distributed Systems and Networks**: To facilitate communication across various platforms, "Draw It or Lose It" can employ a distributed architecture based on microservices. By developing each component of the game—such as game logic, user management, and scoring—as a separate service, The Gaming Room can enhance flexibility and scalability. Each service can communicate over a network using APIs, allowing the development team to update individual components without downtime for the entire application. This independence fosters rapid development cycles and can enhance deployment efficiency. Leveraging cloud services for hosting not only provides high availability and redundancy but also enables the application to scale resources up or down based on real-time demand. However, it's crucial to mitigate potential network dependencies. Implementing load balancers can distribute traffic efficiently, while failover mechanisms can automatically redirect traffic in case of service failure, ensuring users experience minimal disruption.
6. **Security**: Security is a critical concern for protecting user information across various platforms, especially in a gaming context where personal data and payment information may be involved. Linux provides robust security features through user permissions and access controls, which help restrict unauthorized access to sensitive areas of the system. Built-in firewall capabilities also allow for filtering incoming and outgoing traffic, safeguarding the application from various network attacks. To bolster security even further, The Gaming Room should implement strong encryption protocols for data in transit (using TLS) and data at rest (using AES). This ensures that user data remains secure whether it's stored in the database or transmitted between clients and servers. Regular security audits and timely software updates are essential to address and eliminate vulnerabilities proactively. Additionally, employing secure authentication methods—such as OAuth for API permissions or two-factor authentication for user accounts—will help add another layer of protection, ensuring users feel safe and secure while engaging with the game.