

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_gjdgxs)

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 06/22/24 | Kevin Fleming | Project Three |

**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](#_35nkun2)

Creative Technology Solutions (CTS) is partnering with The Gaming Room to develop a web-based version of their popular Android game, Draw It or Lose It. This project aims to create a cross-platform game that allows multiple teams to compete by guessing images rendered from a library of stock drawings. Key requirements include multi-team support, unique identifiers for games, teams, and players, and ensuring only one instance of the game exists in memory at any given time. CTS will handle the setup and configuration of the development environment, design an intuitive user interface, ensure data security, optimize performance, and provide ongoing support post-launch to ensure a seamless and engaging user experience.

## 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](#_1ksv4uv)

Developing the game application in a web-based distributed environment involves several design constraints, including network latency, data consistency, scalability, and security. Network latency cna impact real-time gameplay, necessitating efficient data transmission and minimal lag. Data consistency must be maintained across all distributed components to ensure the game state for each player is accurate. Scalability is important to accommodate varying numbers of users without degrading performance. Security measures are essential to protect user data and prevent unauthorized access, especially given the distributed nature of the application.

## [System Architecture View](#_44sinio)

<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](#_2jxsxqh)

The Entity class serves as a base class for Game, Team, and Player. GameService has a one-to-many association with Game, Game has a one-to-many association with Team, and Team has a one-to-many association with Player. ProgramDriver uses SingletonTester to test the singleton pattern. Encapsulation is evident in the diagram provided as each class encapsulates its attributes and provides methods to interact with these attributes. For example, the GameService class manages games, players, and teams through its methods, hiding the implementation details from other classes. The diagram suggests inheritance through the Entity class, promoting code reuse and organization of related classes. The GameService class is designed as a singleton, ensuring that there is only one instance of GameService throughout the application. This is achieved through the getInstance() method. The relationship between GameService, Game, Team, and Player illustrates aggregation as a GameService contains multiple Game objects, which in turn contains Team objects, and which contains multiple Player objects. This demonstrates how complex systems can be composed of simpler, reusable components.

**"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](#_z337ya)

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** | Unix-based OS with user-friendly interface. Not common for server environments. Advantages are that it’s user-friendly, integration with other Apple products and services are seamless, and it has strong security features. Supports server-based deployment, but options are limited compared to other platforms and higher costs for setup due to need for Apple hardware and macOS Server software. | Open-source and many flavors available for web servers. Strong community support and documentation. Also highly customizable. Advantages are that depending on the OS flavor choice, it’s free, great performance in server environments, regular security updates, and supports a wide range of hardware for easy scaling. Weaknesses are that it can be more complex to set up, and requires technical expertise to manage. Linux systems support various server deployment methods, such as Nginx, Apache, and Docker. | Widely used OS with great support and documentation. Advantages are that it’s user friendly, integrates well with other Microsoft products and services, and offers extensive server software options, such as IIS and Azure. Weaknesses are that there are costs associated with the Windows Server OS, and because of the wide usage, the OS is more frequently targeted by attackers. | Not typically used to host server-side applications. Advantages might be portability. Weaknesses are that it’s not suitable for server-based deployments, not designed to scale or handle large-scale applications, and limited server software options. Again, not suitable for server-based apps. |
| **Client Side** | There is a higher cost for Apple software. Developing on macOS is pretty straightforward as it supports popular text editors and web browsers. Additional time may be needed to ensure consistency across web browsers. Expertise in HTML, CSS, Typescript, and other frontend technologies required. | Excellent choice for development as most flavors support popular IDEs and web browsers, in addition to being very performant for testing code. Minimal cost as Linux is open-source and other tooling needed for development is generally free. Additional time may be needed to ensure consistency across web browsers. Expertise in HTML, CSS, Typescript, and other frontend technologies required. Familiarity with Bash, and the Linux command line can be beneficial. | Cost incurred for Windows OS. Support for popular IDEs and web browsers, although setup can sometimes differ from macOS and Linux which I’ve found to be generally more straightforward. Additional time may be needed to ensure consistency across web browsers. Expertise in HTML, CSS, Typescript, and other frontend technologies required. Familiarity with PowerShell and CMD Prompt can be beneficial. | Mobile OSs support web standards and mobile-specific optimizations, and have development frameworks such as React Native and Flutter to build for mobile clients. Development tools such as Xcode and Android Studio are free, but testing on real devices can incur additional costs. The Apple Developer Program costs $99 per year. Deployment solutions such as Expo and EAS have costs as well, but generally eases the pain of delivering applications for multiple mobile platforms from one code framework. Mobile development requires additional time for optimizations and testing across different screen sizes, and ensuring responsive design. Expertise in HTML, CSS, Typescript, and other frontend technologies required. |
| **Development Tools** | Visual Studio Code, Xcode, iOS Simulator, Android Studio, Web browsers, Web browser Developer Tools, Homebrew (package manager), node, npm, git, React, React Native, TypeScript/JavaScript, HTML5, CSS3, Material UI, shadcn/ui, Github, .NET Core, C#, Docker | Visual Studio Code, iOS and Android emulators, Web browsers, Web browser Developer Tools, apt-get/yum/etc… (package manager for specific Linux flavor), node, npm, git, React, React Native, TypeScript/JavaScript, HTML5, CSS3, Material UI, shadcn/ui, Github, .NET Core, C#, Docker, Apache, Nginx | Visual Studio, Visual Studio Code, iOS and Android emulators, Web browsers, Web browser Developer Tools, node, npm, git, React, React Native, TypeScript/JavaScript, HTML5, CSS3, Material UI, shadcn/ui, Github, .NET Core, C#, Docker, IIS | Visual Studio Code, iOS and Android emulators, Web browsers, Web browser Developer Tools, node, npm, git, React, React Native, TypeScript/JavaScript, HTML5, CSS3, Material UI, shadcn/ui, Github, .NET Core, C#, Docker, Flutter, Dart |

## 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**: Based on the previous evaluation, I recommend using Linux as the primary platform for expanding The Gaming Room’s “Draw It or Lose It” game. Linux offers several advantages that align well with the project’s requirements and design constraints. Linux is highly customizable and supports a wide range of server deployment methods, such as Nginx, Apache, and Docker, making it ideal for a distributed environment. Its open-source nature means that there are no licensing costs, which can significantly reduce the overall expenses associated with server setups and maintenance. Additionally, Linux is known for its robust performance and regular security updates, ensuring that the game operates smoothly and securely, even under high traffic conditions. Linux has strong community support and extensive documentation make it a reliable choice for development and troubleshooting. The platform's compatibility with popular development tools and IDEs, such as Visual Studio Code, Android Studio, and various package managers, ensures that developers can work efficiently and effectively. The minimal cost and high performance of Linux in testing and development environments further enhance its appeal.
2. **Operating Systems Architectures**: Linux is an open-source Unix-like operating system, known for its robustness, flexibility, and customization capabilities. It supports various distributions, such as Ubuntu, CentOS, and Debian, each offering a solid foundation for server environments. The customization and flexibility of Linux allow for optimized performance and resource utilization based on the specific needs of The Gaming Room. This is particularly useful for creating a scalable and efficient web application, as Linux supports various server deployment methods, including Nginx, Apache, and Docker. In particular, Docker facilitates containerization, which allows an application to be packaged with all its dependencies, ensuring consistent performance across different environments. On the client-side, Linux supports a wide range of development tools, such as Visual Studio Code, Android Studio, and various web browsers, making it an excellent platform for developing and testing web-based applications. The support for popular programming languages and frameworks like JavaScript, React, and Docker ensures that developers have all the necessary tools for building and maintaining the application. This support for both the server-side and client-side ensures a seamless and engaging user experience, fulfilling The Gaming Room’s requirements effectively.
3. **Storage Management**: An appropriate storage management system for the Linux operating system is MySQL. MySQL is an open-source relational database management system that offers robust performance, scalability, and ease of integration with web applications. It supports ACID-compliant transactions, ensuring data consistency and reliability, which are important for a distributed gaming application like "Draw It or Lose It." MySQL's compatibility with various Linux distributions and its extensive community support make it a reliable choice for managing the game's data. Additionally, MySQL's replication and clustering features provide high availability and fault tolerance, essential for maintaining uninterrupted service in a distributed environment. Its ability to handle large volumes of data make MySQL an ideal storage management solution for The Gaming Room's expansion plans.
4. **Memory Management**: Linux uses memory management techniques that are highly beneficial for running “Draw It or Lose It”. It uses a virtual memory system that allows for efficient allocation and management of memory resources, which ensures that the application will run smoothly even under varying loads. Linux uses paging and swapping to manage physical memory, which ensures that the active data remains in RAM while less frequently accessed data is moved to swap space. The Linux kernel uses caching and buffers to help reduce disk Input/Output operations, enhancing performance. These memory management techniques ensure that "Draw It or Lose It" operates efficiently, with minimal latency and high reliability, even as the number of concurrent users increases.
5. **Distributed Systems and Networks**: To enable communication between various platforms for “Draw It or Lose It”, RESTful APIs can be implemented. RESTful APIs allow different parts of the application to communicate over HTTP, providing a platform-independent method for exchanging data. For real-time communication, WebSockets can be used to provide real-time, bidirectional communication, which is essential for the game's interactive nature. A microservices architecture can be adopted, where different functionalities of the game, such as user authentication, game logic, and scoring, are handled by separate services. Each service can be independently deployed, scaled, and maintained, enhancing the overall flexibility and reliability of the application. To ensure seamless connectivity and handle potential outages, the use of load balancers is essential. Load balancers distribute incoming traffic across multiple servers, ensuring no single server becomes a bottleneck. Dependencies between components can be managed using container orchestration tools like Kubernetes, which ensure that services are automatically restarted in case of failures and can scale based on demand. This approach ensures that "Draw It or Lose It" remains accessible and performs optimally across various platforms and network conditions.
6. **Security**: To protect user information while using Linux, HTTPS communication should be enforced with TLS encryption to secure data transmission. This will ensure that all data exchanged between a server and a client is encrypted. A robust authentication mechanism, such as OAuth 2.0, should be used to manage user access and prevent unauthorized access. Data at rest should be encrypted using advanced encryption standards (AES), and sensitive information should be stored securely, using hashing algorithms for passwords. Regular security updates and patches are of the utmost importance. This will ensure The Gaming Room’s Linux instances and software components are up-to-date in their defense against vulnerabilities. Implement firewall rules and intrusion detection systems to monitor and control incoming and outgoing network traffic. Security tools available in the Linux ecosystem, such as SELinux or AppArmor, can be utilized to enforce strict access controls and isolate applications. Finally, conduct regular security audits and penetration testing to identify and address potential security weaknesses, ensuring the continuous protection of user information across all platforms.