

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 | 01/23/25 | Alexander Hill | Initial summary, descriptions, and requirements |

**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 well-known game Draw It or Lose It from The Gaming Room will be converted from an Android-only app to a cross-platform, web-based application that can be played on PCs, tablets, and smartphones by Creative Technology Solutions (CTS). The new approach will provide a smooth and synchronized user experience by supporting numerous teams, distinct game identifiers, and a single instance of the game in memory. CTS will provide a reliable and future-proof solution that satisfies The Gaming Room's requirements by utilizing scalable architecture and contemporary web technologies. As part of our process, we walk the client through setting up the environment and using best practices to make deployment easier. With this update, The Gaming Room will be better positioned to grow in the cutthroat gaming industry, reach a wider audience, and enhance user experience.

## Requirements

Business Requirements

* Expand the Draw It or Lose It game to a cross-platform, web-based application accessible on multiple devices (desktops, tablets, and smartphones).
* Maintain the core gameplay mechanics and user engagement of the original Android app.
* Ensure that game and team names are unique.
* The game must consist of 4 one-minute rounds.
* If a puzzle is not guessed in time, the remaining teams each have one chance to guess with a 15 second time limit.
* Drawings must be rendered steadily, reaching completion at 30 seconds.
* Support one or more teams with multiple players assigned to each team.
* Provide a streamlined deployment process and clear documentation for ease of setup by non-technical staff.

Technical Requirements

* Enable cross-platform compatibility through technologies like HTML, CSS, and JavaScript.
* The game must be capable of functioning with one or more teams, with teams composed of at least 2 individuals.
* Ensure unique identifiers for games and teams using a backend database with constraints for uniqueness.
* Allow only one instance of the game to exist in memory at a time through unique identifiers.
* Ensure scalability and performance to handle varying levels of traffic.

## [Design Constraints](#_2et92p0)

* Cross-Platform Compatibility: The application must run seamlessly on various devices and operating systems, requiring responsive design and extensive testing across browsers and devices.
* Unique Identifiers: All games, teams, and players must have unique identifiers to prevent duplication, necessitating robust backend database constraints and logic.
* Single Game Instance: Only one instance of the game can exist in memory at any time, requiring centralized state management and efficient real-time synchronization.
* Scalability: The application must be able to handle varying traffic loads across all versions of the application.
* Security: As for any web-based application, user data and communication must be secure and encrypted.
* Real-Time Synchronization: Gameplay events, as in the rendering of drawings, must be synchronized across all users in real time.

## [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 illustrates The Gaming Room's game application using object-oriented principles. The Entity superclass demonstrates both abstraction and inheritance as it provides shared attributes (id and name) and methods (getId(), getName(), and toString()). These are inherited by the Game, Team, and Player classes, reducing redundancy and ensuring consistency. In terms of abstraction, by focusing on the essential features for all the subclasses, Entity can hide these unnecessary implementation details and provide a simple interface to use. The GameService class implements the Singleton design pattern, ensuring only one instance exists to manage all game-related operations, fulfilling the requirement of having a single instance of the game in memory at any time. The Game class manages a list of Team objects, and the Team class manages a list of Player objects, supporting multi-team functionality with unique identifiers. Encapsulation is highlighted by many of the class attributes being marked as private, ensuring that they cannot be accessed from outside the class. Polymorphism is demonstrated by the overridden toString() method in subclasses, allowing tailored representations for Game, Team, and Player. This design ensures scalability, maintainability, and efficient development, meeting the client's technical and business requirements for the web-based application.

**"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** | Although it is not a top option for web hosting, MacOS provides server-based deployment alternatives through third-party solutions like Apache, Nginx, and Docker. Its acceptance for enterprise hosting is constrained by the lack of industry-standard tools and the higher cost of hardware, despite the fact that it offers a reliable and secure Unix-based architecture and seamless connection with Apple services. Furthermore, the cost is increased because macOS may only be installed lawfully on Apple devices. Buying Apple equipment for hosting makes it a costly alternative, even if the operating system itself does not have a separate licensing charge. | The most popular web hosting platform is Linux, which supports cloud-based services like AWS, Azure, and Google Cloud as well as a number of server-based deployment alternatives like Apache, Nginx, and MySQL. Because it is free to use, open-source, and extremely adaptable, it appeals to budget-conscious companies. It is also the recommended operating system for web servers due to its stability, security, and robust community support. However, certain distributions require more configuration, and Linux has a steeper learning curve for people who are not accustomed with command-line operations. Enterprise versions like Red Hat and SUSE may have related maintenance fees, but the majority of distributions, including Ubuntu, Debian, and CentOS, are free. | A stable hosting environment is offered by Windows Server, especially for applications that use Microsoft technologies like MSSQL, ASP.NET, and .NET. It works nicely with Microsoft's cloud-based products like Azure and supports IIS (Internet Information Services) for site hosting. Compared to Linux, Windows Server is easier to administer since it provides enterprise-level support and an intuitive graphical user interface. However, it uses a lot of resources and typically needs more upkeep to guarantee performance and security. Windows Server is a more costly option than Linux for hosting a web-based application because licensing fees vary, usually ranging from a few hundred to several thousand dollars. | Although they are not appropriate for hosting web-based apps, mobile devices (iOS, Android) are vital user access points. Mobile platforms give consumers access through browsers or mobile-friendly apps and are based on web servers that run on different operating systems. Although they provide for more accessibility and reach, mobile devices are not viable for direct hosting due to their limited processing power, storage capacity, and network stability. Since mobile devices act as clients accessing server-hosted or cloud-based web applications rather than hosts themselves, there are no direct license fees involved with hosting on them. |
| **Client Side** | Knowledge of the macOS system architecture is necessary for MacOS development, especially if the application connects with services that are exclusive to Apple. Additional testing and optimizations could be necessary to ensure compatibility with Safari, the default web browser, particularly for features that rely on WebKit. Because testing and development on macOS require Apple gear, the cost of development might be considerable. Dedicated quality assurance and possible UI element modifications are necessary to guarantee flawless operation for macOS users, even though macOS is not frequently utilized as a major hosting platform. If macOS-specific optimizations are required, development time may increase. | Linux is open-source and cost-effective for hosting, requiring standard browser testing for Linux clients as most web applications operate on Linux servers. The key focus is ensuring compatibility with Chrome, Firefox, and Edge on supported Linux versions. Development times are often shorter due to stable environments, but expertise in Linux servers, scripting, and security is essential. Containerization tools like Docker and Kubernetes can simplify deployment and improve scalability. | Windows needs to make sure that it works with Chrome, Microsoft Edge, and Internet Explorer (for legacy computers), which could take more time to create and test. Because Windows is used by many business users, it is crucial to optimize the application for both older and newer versions of the operating system. Using Microsoft-specific tools, such ASP.NET and IIS, might raise development expenses because they require additional licensing fees. Maintaining application stability requires knowledge of Windows system administration and security procedures. The development period may also be extended by testing across several Windows versions and making sure the application is responsive at various screen resolutions. | Mobile devices like iOS and Android create challenges for cross-platform compatibility. To achieve a smooth experience, it is essential to have touch-friendly interfaces, flexible layouts, and responsive design. Mobile browsers can access web-based applications, but they need further testing to work well on Safari, Chrome, and others. Performance varies because of differences in processing power and network capabilities. Developing a native mobile app could increase time and costs, requiring knowledge of Kotlin or Java for Android and Swift for iOS. Alternatively, Progressive Web Applications (PWAs) can simplify development while ensuring a mobile-friendly interface. |
| **Development Tools** | For native applications, MacOS development mostly uses Swift and Objective-C, with Xcode serving as the main IDE. Standard web technologies like HTML, CSS, and JavaScript, as well as frameworks like React or Angular, can be used to create web-based applications for MacOS users. To guarantee compatibility, more testing of Safari's WebKit engine could be necessary. The impact on the development team is minimal if cross-platform web technologies are used, but dedicated MacOS developers may be required for specific integrations. Xcode is free, but Apple Developer Program membership ($99/year) is necessary for app distribution on the App Store. | Linux development is well-supported by open-source technologies. Python, Java, PHP, and JavaScript are common programming languages, and well-known IDEs include Visual Studio Code, JetBrains IntelliJ, and Eclipse. MySQL or PostgreSQL databases running on Apache or Nginx servers are frequently used by Linux-based web applications. Development time is decreased by Linux's powerful command-line tools, which facilitate automation and deployment. The majority of web development teams already have the requisite knowledge because Linux is frequently used for server hosting. There are no licensing costs for Linux itself, but some enterprise tools (e.g., Red Hat, JetBrains products) may require paid subscriptions. | Visual Studio is the main IDE for Windows development, which also uses Python, JavaScript, and.NET (C#, ASP.NET). Many enterprise apps interface with Microsoft SQL Server, and Windows Server provides IIS for hosting web applications. More testing may be necessary to ensure compatibility across various Windows versions. Windows-specific knowledge may be required by development teams, particularly for corporate or legacy system integrations. Professional editions and Windows Server licenses cost money, however Visual Studio Community is free. Additional cost considerations depending on resource utilization apply if cloud deployment is done via Microsoft Azure. | With Xcode and Android Studio serving as the main IDEs, native development on mobile devices (iOS and Android) necessitates proficiency in Swift for iOS and Kotlin or Java for Android. There is less need for distinct teams thanks to cross-platform frameworks like Flutter, React Native, and Xamarin, which enable a single codebase. Mobile browsers such as Chrome (Android) and Safari (iOS) must be used to test web applications. Because mobile development requires more testing and device compatibility, it can be more expensive. Google Play and the Apple App Store charge yearly developer fees ($25 for Google Play, $99 for Apple), whereas Xcode and Android Studio are free. |

## 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 growing Draw It or Lose It, Linux—more especially, a cloud-based distribution like Ubuntu Server—is the suggested server platform. Linux offers broad compatibility with web-based applications, high scalability, and cost-effectiveness. It is also the best option for a distributed gaming environment due to its security features and broad support for web hosting technologies.
2. **Operating Systems Architectures**: My suggested architecture uses a client-server model, with users interacting through web clients or mobile devices and the game logic and data processing taking place on the server. To guarantee effective resource allocation and scalability, the application will be orchestrated with Kubernetes and containerized with Docker, which I discussed in the client-side evaluation of Linux. Several gaming features, including user identification, game session management, and real-time communication, can be modularized using a microservices design.
3. **Storage Management**: For effective storage of game states, user profiles, and game data, a cloud-based relational database such as PostgreSQL or MySQL is advised. NoSQL technologies like MongoDB can be utilized for session management and caching frequently accessed data to improve performance and scalability. Game assets like pictures and media files can be stored on cloud storage services like Amazon Web Services or Google Cloud Storage.
4. **Memory Management**: Virtual memory paging, which optimizes resource allocation by moving dormant processes to disk storage as needed, is how the Linux server will handle memory. By keeping frequently used game data in RAM, memory caching techniques can be used to lower latency. Additionally, the game server can benefit from effective memory management through the use of Garbage Collection (GC) techniques in Python or Java.
5. **Distributed Systems and Networks**: Multiplayer gameplay will be seamless thanks to real-time communication between clients and the game server made possible by a RESTful API. Incoming traffic will be divided across several instances of the game server by the distributed architecture using load balancing strategies like NGINX, as discussed previously in the server-side discussion of Linux. By optimizing asset loading times, cloud-based CDNs (Content Delivery Networks) will lower latency for users in various geographical areas. During outages, availability will be maintained through the use of redundancy procedures and failover techniques – for example, replication of the primary database as well as automated backups of data.
6. **Security**: End-to-end encryption will be used to ensure secure connection between clients and servers and in turn protect user data. To provide safe access to the game, I am recommending an authorization framework like OAuth 2.0 for user login and session management. Permissions inside the system will be controlled by Role-Based Access Control (RBAC) – this is essentially a security model which restricts system access based on a user’s role within the application or organization. Potential cyberthreats will be reduced by firewalls, intrusion detection systems (IDS), and frequent security assessments. DDoS defenses will also be put in place to stop malicious attempts from overloading servers.