

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 | 07/20/25 | Kayla Rose | Initial draft completed, includes project overview, requirements, design constraints, UML analysis, platform evaluation, recommendations, and conclusion. |

**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 has requested support from Creative Technology Solutions in order to expand their current Android-based game, Draw It or Lose It, into a cross platform, web-based application. Their current application is limited to Android devices, but this is restricting the game’s reach and user accessibility. With the lack of centralized and scalable design it also adds the challenge of managing game sessions, teams, and players in the most efficient way. The existing implementation also does not enforce any uniqueness of game or team names with increases the risk of any data inconsistency or conflicts.

To best address these issues, CTS is proposing the development of an object-oriented software design that supports a multi-team and multiplayer game environment accessible through a web interface. This will include:

* **Singleton pattern:** This is to ensure that only one active game instance exists in memory at any given time to improve the performance of the application and prevent any conflicts.
* **Unique identifiers:** For each game, team, player there will be unique identifiers to maintain data integrity and simplify tracking.
* **Unique naming conventions:** There will be logic to enforce unique names for teams and games to enable players to check for the availability of a certain name during their account creation or joining process.
* **Scalability:** This will have scalable architecture that supports any future expansions that may include potentials deployments on additional platforms. This could be IOS or desktops or others.

This design lays the foundation for a centralized game system that has the ability to accommodate a broader audience all while preserving the core gameplay experience.

## Requirements

***Business Requirements:***

* ***Multiplatform accessibility:*** *Develop a web-based version of the existing mobile game to support multiplatform.*
* ***Retain the core structure:*** *We will need to retain the core game-play structure to include four one minute rounds, image rendering timelines, and team-based guessing mechanics.*
* ***Multi team:*** *Need to allow for multiple teams per game and multiple players per team****.***
* ***Unique Player/Team names:*** *Ensure that the system has the ability to verify is the name of their choice has already been taken either for their player or team names.*
* ***Multiplayer capabilities:*** *Support for the game to expand beyond Android users and host online multiplayer experience.*

***Technical Requirements:***

* ***Singleton Design Pattern:*** *Implement this pattern to ensure that there is only one instance of the game in memory at any given time.*
* ***Unique Identifiers****: Assign IDs to every game/team/player to support any internal tracking and management.*
* ***Scalability:*** *Allow for future updates that include potential expansions to other platforms and devices.*
* ***Real time synchronization:*** *Enable real time synchronization and interaction between the users during their gameplay.*
* ***Cross browser compatibility:*** *Ensure that the game is hosted in an environment that supports this functionality and modern web standards.*

## [Design Constraints](#_2et92p0)

When we are developing Draw It or Lose It as a web based application that introduces several design constraints that we need to consider during planning and development. These constraints are a direct effect to the architecture, performance, scalability, and user experience.

* Singleton Game Instance Constraint: Only on instance of the game should exist in the memory at any given time. By implementing a singleton design pattern we can ensure a controlled and consistent game management. When this is used in a web environment this constraint can create challenges related to stat synchronization across sessions, especially if hoisted on multiple servers (Alon, 2025).
* Unique Identifiers for Games, Teams, and Players: There must be unique identifiers to prevent any collisions and ensure accurate tracking. This creates a constraint on the backend architecture which has to be capable of generating, storing, and validating the unique identifiers in real time (Usool, 2024). If there is a failure to enforce this constraint then it can result in data overlap, session errors, or misassigned players.
* Web based and cross platform compatibility: The application must function across multiple devices and browsers so all components including their user interface and backend applications must comply with web standards. They also need to be tested for cross browser compatibility. This constrains the use of certain platform-specific features and requires a careful UI/UX design to ensure there is consistent behavior or responsiveness across all devices (Dhillon, 2020).
* Real Time interaction and low latency: There is a requirement for real time data processing and low latency communication because the game involves timed rounds with live player interaction. This constraint impacts technology choices and requires some sort of bidirectional communication between the client and server (Samba, 2024). This also requires robust handling error and session management in order to account for issues that may arise. This could be player disconnects, delays, or even server failures.
* Scalability and Resource Management: Since the game has to support multiple concurrent teams and players there is a constraint on the system’s ability to be efficiently scalable. The developers will need to then design a system that can handle any loads without affecting the performance. This could include load balancing, cloud infrastructure, or smaller deployment to scale horizontally (Estrach, 2023).
* Security and Data Validation: With the addition of the users being able to create and join games via the web then the system needs to reinforce input validation, secure authentication, and session protection (Ojha, 2024). This is a constraint on the development by requiring security best practices throughout the codebase and could impact how user input, team names and sessions are managed to avoid exploits.

These constraints shape both the design and implementation phases of development. We can address then early to ensure that the game is reliable, secure and performs well in a real world environment.

## [System Architecture View](#_ilbxbyevv6b6)

Not a current requirement for this phase of the project. This is reserved for detailing the system and subsystem architecture of the game application. In future applications this will include further descriptuions of:

* The tiered architecture
* The logical communication topology
* Physical components
* Data flow between users, application logic, and storage

## [Domain Model](#_8h2ehzxfam4o)

The provided UML diagram illustrates the structure of the gaming application for The Gaming Room. This shows how the classes interact and fulfill core software requirements while using object-oriented programming principles.

Class relationships:

* Entity: This is a base class for *Game, Team,* and  *Player.*  There are shared attributes such as *id*  and *name.*  There are common methods such as  *getId(), getName(),* and *toString().*
  + This is an example of inheritance because it promotes code reuse and reduces any duplication across similar classes.
* GameService: This is our singleton class that manages the overall game environment. It holds:
  + A list of *Game* objects
  + Counters for next IDs – (*nextGameId, nextTeamId, nextPlayerId)*
  + Methods for game management – *addGame(), getGameCount()*
    - This implements the use of a singleton pattern and ensures that only one instance of the game service exists in memory which fulfills out clients requirement.
* Game: Contains a list of *Team* objects and supports the addition of teams via *addTeam().*
  + This helps to show a composition relationship meaning 1 game to 0..\* Teams. Allowing for each game to manage multiple teams.
* Team: This constains a list of *Player* objects and supports the addition of players via *addPlayer().*
  + This also shows composition with the same relationship of 1 team to 0..\* Players.
* Player: Inherits from *Entity* and represents individual users in the game.
* ProgramDriver and SingletonTester: These classes are used to test the behavior of the application. *ProgramDriver* has the *main()* method to launch the app while the *singletoTester* verifies that *GameService*  is behaving as the intended Singleton.
  + The <<uses>> relationship indicates that *programDriver* and *SingletonTester*  interact with the service layer.

There are multiple ways OOP Principals are being demonstrated here.

1. Encapsulation: Each class has its own fields and methods to manage their internal state. We can see this in GameService as it hides its list of games but exposes methods in order to manipulate them
2. Inheritance: *Game, Team,* and *Player* extend the *Entity* class. This in turn allows them to share common behaviors and attributers without needing to change the code.
3. Abstraction: *Entity*  used as a general class for named objects abstracts shared behavior, simplifying the design.
4. Singleton Pattern: Implemented in *GameService* using *getInstance()* method. This is to make sure only one instance of the class exists during the application’s runtime which also fulfills the requirement that only one game exists in memory at any given time.

This supports the software requirements in many ways. We are supporting the requirement of multiple teams and players per game in our *Game* and *Team* classes. We also satisfied the unique ID requirement as we handle this in our *GameService* class. The unique names are enforced with name checking methods that are implemented in *addGame* and *addTeam.* Finally, the singleton constraint is enforced through the entirety of the structure of *GameService.*

**"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 can be used for web-based hosting but is not normally preferred for server deployment in production environments. macOS shares a Unix foundation with Linux, making it pretty stable and secure, but it is limited by hardware dependency and higher costs (Pranto, 2024). It is more commonly used for local development and testing than as a dedicated web server. | Linux is the more dominant OS for hosting web-based applications. It is open-source, lightweight, highly customizable, and widely supported by web technologies. It offers strong performance, robust security, and low cost. A major advantage is that most cloud service providers offer native Linux-based deployments (educative, n.d). | Windows servers are mostly used in enterprise environments, more so when .NET or Microsoft SQL Server technologies are involved (educative, n.d). However, they come with licensing fees and are generally heavier on system resources. For web-based applications using open technologies Windows is functional but not as much as Linux. | Mobile devices are not designed to act as servers. Apps can use a mobile device for peer-to-peer communication or limited hosting but they lack the stability, power, and networking capabilities required for full server-side web application hosting (Vault, 2023). |
| **Client Side** | Supporting Mac clients requires ensuring that the web app functions well on Safari, and the UI must account for differences in macOS input behavior. Macs are common in creative and academic fields, so testing should include this platform. Development cost is moderate, as most tools are cross-platform, but access to macOS hardware is essential for testing (Roell, 2024). | Linux desktop clients are less common among general users but prevalent among developers. Supporting Linux requires testing in Firefox and Chromium-based browsers. Cost is minimal due to open-source tools, but time and expertise are needed to handle compatibility nuances in window managers or font rendering (Roell, 2024). | Windows is the most widely used desktop platform, especially for games and casual users. Supporting Windows clients is essential. Testing must include Edge and Chrome browsers, and user interface responsiveness for different screen resolutions. Development is straightforward and cost-effective due to the wide availability of tools and testing environments (Roell, 2024). | Mobile client support is critical, as users increasingly access games via mobile browsers and apps. Development must focus on responsive web design, touch interaction, and device fragmentation (iOS vs Android, screen sizes). Cost and time are higher due to the need for mobile testing labs or emulators. Expertise in mobile-first design is crucial. |
| **Development Tools** | Development for Mac can be done using Xcode, Visual Studio Code, and terminal-based tools. Languages like Swift (native Mac), Python, JavaScript, and Java are commonly used. macOS is Unix-based, so it supports many Linux-style tools. It’s well-suited for frontend development and cross-platform web apps (tar, 2023). | Linux supports a vast array of open-source tools including Eclipse, VS Code, Vim, and JetBrains IDEs. Languages such as Python, JavaScript, Ruby, Java, and Go are popular. Docker, Git, and CI/CD tools are well-integrated. Linux provides an ideal development environment for backend services, server setup, and testing (Tar, 2023). | Windows developers often use Visual Studio, VS Code, and .NET-based tools. It supports a wide variety of programming languages including C#, JavaScript, Python, and Java. It's also compatible with Docker, Node.js, and other web development stacks. Windows is a practical environment for full-stack development but requires more setup for Unix-based workflows (Tar, 2023). | Mobile development requires tools like Android Studio, Xcode (for iOS), Flutter, and React Native. Languages include Java, Kotlin, Swift, and Dart. Developers also use web-based tools for responsive design and testing, such as Chrome DevTools. Mobile dev has a steep learning curve, especially when developing native apps across platforms, but frameworks like React Native reduce redundancy (Tar, 2023). |

## 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 Draw It or Lose It, the expansion into a multi platform web based game I am recommending that the operating platform is Linux. Linux is the industry standard for hosting scalable, secure, and cost effective web applications. It is very compatible with modern frameworks and supports all of the major programming languages such as Python or JavaScript. There is also good integration with cloud platforms such as AWS or Google Cloud. As far as client access the platform should be browser based in order to ensure compatibility across windows, macOS, Linux, Android, and IOS.
2. **Operating Systems Architectures**: The Linux operating system architecture follows a modular and layered design. There is a monolithic kernel that manages system resources like memory, devices, and processes. There is support for multi-user and multitasking capabilities which is essential for managing multiple game sessions or players at the same time. There is also support. For virtualization and containers with Linux, which enables rapid deployment. Sandboxing and scalability of application instances. There is an open-source nature that allows for customization and optimization specific to the performance needs of the game server.
3. **Storage Management**: As far as storage management, cloud based storage is the recommended solution. As it provides a scalable, secure and redundant data storage. This is ideal for storing user accounts, gameplay, sessions, images and static assets required by the game. In addition, a relational database. Hosted on the Linux server should be used for storing structured game data such as teams, players or scores. These databases integrate seamlessly with Linux and offer important yet powerful querying, indexing and backup capabilities.
4. **Memory Management**: Linux also uses an advanced virtual memory management system. This includes paging, segmentation and swap space in order to optimize their performance. When running draw it or lose it, the OS allocates physical and virtual memory to the game instance, which manages active data in RAM while moving less used data. To swap if needed. The Singleton pattern. We used in our game design helps limit the number of in memory objects to ensure efficient use of memory. Garbage collection and memory monitoring tools can help maintain any optimal performance without any memory leaks or over consumption.
5. **Distributed Systems and Networks**: In order to best support gameplay across devices and platforms, draw it or lose it should be deployed as a distributed web application that uses a client server model. The server hosted on a Linux environment. Handles game logic, data storage and session management. Clients can communicate with the server. Using a number of different applications such as Restful Apis or Websockets for real time interaction. We can also use load balancers, microservices and container orchestration. To allow the system to scale horizontally and manage fault tolerance. Distributed components must be able to handle dependencies such as a session state, user authentication, and data synchronization. By implementing fall backs and retry logic, we can address connectivity issues or service outages to ensure a smooth gameplay experience even under network strain.
6. **Security**: Security is one of the most important aspects. As we are developing and implementing this project. The following measures are a necessity for implementation to protect user information across platforms:
   1. Data Encryption: The use of HTTPS to encrypt all data in transit and the ability to encrypt sensitive data e at rest using AES-256 encryption provides the required security.
   2. Authentication and Authorization: We must use secure authentication protocols such as Oauth 2.0 and implement role based access control to ensure that players teams and admins have the Appropriate access to the respected user.
   3. Input Validation and Sanitization: This prevents injection attacks by validating all client inputs.
   4. Session management: Using secure session tokens with expiration and renewal strategies by also storing session data server side or in a secure memory.
   5. Monitoring and logging: We must implement intrusion detection logging and monitoring tools to track any unusual activity.
   6. Platform Hardening: This will keep all server software up to date disable unused ports and use a firewall in order to restrict any unauthorized access.

By using all of these layered security measures alongside Linux’s internal performance system game can safely support players across all target platforms.

With the expansion of Draw It or Lose It turning into a web based cross platform game it presents a great opportunity for the gaming room to reach a Broader audience while enhancing user engagement. By choosing a Linux based server a robust object-oriented design with a secure and scalable technology, this program can meet both the current gameplay requirements as well as any future growth needs. In our project we have outlined a clear strategy for managing game instances teams and players efficiently While at the same time ensuring seamless communication between diverse platforms. We were also able to pay careful attention to system architecture, storage memory and security and our proposed solution to the gaming room for delivering reliable and engaging gaming experience across desktop and mobile applications. Moving forward continued testing and user feedback will be important for refining the application and ensuring long term success.

References:

Digital Samba. (n.d.). *WebSocket vs HTTP*. Digital Samba. <https://www.digitalsamba.com/blog/websocket-vs-http>

Educative. (n.d.). *Differences between Windows, macOS, and Linux operating systems*. Educative.io. <https://www.educative.io/answers/differences-between-windows-macos-and-linux-operating-systems>

Ojha, H. (n.d.). *Secure game authentication: Encryption techniques and their role in Unity game development*. Medium. <https://medium.com/@hardikojha079/secure-game-authentication-encryption-techniques-and-their-role-in-unity-game-development-74e4e7075c04>

Pranto, N. (2021, December 8). *MacOS web development setup guide*. DEV Community. <https://dev.to/npranto/macos-web-development-setup-guide-345p>

Sugar Lane Design. (n.d.). *State management in full-stack apps: Synchronizing frontend and backend states*. Sugar Lane Design. <https://sugarlanedesign.com/state-management-in-full-stack-apps-synchronizing-frontend-and-backend-states/>

Tag Vault. (2023, November 7). *Android vs iOS vs Windows vs Linux*. <https://tagvault.org/blog/android-vs-ios-vs-windows-vs-linux/>

Tar, E. (2023, February 10). *What’s the best OS for developers: Mac vs Windows vs Linux*. Medium. <https://medium.com/adroit-group/whats-the-best-os-for-developers-mac-vs-windows-vs-linux-4f93d5565671>

Roell, J. (2024, May 13). *Mac vs Windows vs Linux: The ultimate guide for software developers*. Stackademic. <https://blog.stackademic.com/mac-vs-windows-vs-linux-the-ultimate-guide-for-software-developers-48650d4b5170>

Stanton Street. (n.d.). *What is cross-browser and multi-platform capability?*. Stanton Street. <https://www.stantonstreet.com/blog/what-is-cross-browser-and-multi-platform-capability/#:~:text=When%20an%20application%20is%20programmed%20to%20work%20on,in%20Chrome%2C%20but%20may%20look%20terrible%20in%20Firefox>.

Usool Data Science. (n.d.). *Understanding UUIDs: A backend engineer's guide for junior developers*. DEV Community. <https://dev.to/usooldatascience/understanding-uuids-a-backend-engineers-guide-for-junior-developers-5075>