

Draw It or Lose it Game

# **CS 230 Project Software Design Document**

Version 1.1

## Table of Contents

[**CS 230 Project Software Design Document** 1](#_Toc151834673)

[Table of Contents 2](#_Toc151834674)

[Document Revision History 2](#_Toc151834675)

[Executive Summary 3](#_Toc151834676)

[Requirements 3](#_Toc151834677)

[Design Constraints 3](#_Toc151834678)

[System Architecture View 4](#_Toc151834679)

[Domain Model 4](#_Toc151834680)

[Evaluation 6](#_Toc151834681)

[Recommendations 7](#_Toc151834682)

References 9

## [Document Revision History](#_grjogdjh5fi8)

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| 1.0 | 11/01/2023 | Gonzalo Patino | Released |
| 1.1 | 12/08/2023 | Gonzalo Patino | Modified the recommendations session |

**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)

This document outlines the software design strategy for transitioning “Draw It or Lose It” from an Android application to a web-based platform, accommodating multiple teams and players. My solution addresses the challenge of creating a unified, interactive experience across various devices. I will implement a system where each game and team name are unique and enforce a design that allows only one instance of the game in memory to maintain data integrity and provide a seamless user experience. As I progress, it’s vital for The Gaming Room to engage with the development milestones and provide timely feedback.

## Requirements

*The client’s requirements are as follows:*

1. **Multi-team Capability**: The game must support multiple teams within a single game instance.
2. **Team Composition**: Each team should be able to have multiple players.
3. **Uniqueness of Identifiers**: Both game and team names must be unique to prevent user confusion and conflicts within the game database.
4. **Singleton Game Instance**: The application must ensure that only one instance of the game exists in memory at any given time, using unique identifiers for games, teams, and players.

## [Design Constraints](#_2et92p0)

1. **Cross-Platform Compatibility:** The application must operate seamlessly across browsers and devices; having a responsive design and careful consideration of different web standards and testing across platforms to ensure compatibility.
2. **Network dependencies:** A distributed web-based environment heavily reliant on network stability and speed. A design must be considered for efficient data transfer to minimize latency and ensure the game remains playable even with varying network conditions.
3. **Concurrency Control:** With multiple teams and players interacting in real-time, managing concurrent operations is critical; the system must handle simultaneous requests without data corruption.
4. **State Management:** Ensuring that only one instance of the game is in memory at a time implies using a special design approach just like the singleton.
5. **Unique Identifier Management:** The uniqueness of game and team names requires a robust method for checking and validating identifiers against existing ones.
6. **Scalability:** As the user base grows, the system must scale accordingly without degradation in performance. This involves designing a backend that can handle an increasing number of concurrent users and data volume.
7. **User Experience:** Despite the technical complexities, the application must remain intuitive and engaging for the user. This constraint demands a user-centric design approach with regular feedback cycles.

## [System Architecture View](#_ilbxbyevv6b6)

For the web-based version of “Draw It or Lose It”, I propose a three-tiered architecture that logically separates the front end, business logic, and data storage. This design not only aligns with clients’ requirements but also ensures a scalable and maintainable system that can evolve with future enhancements.

1. **Presentation layer (Client Tier):** This is the front-end or user interface of the game. It will be accessible via web browser using various electronic devices. It is worth mentioning that this presentation layer will be built using responsive design features that work well with different screen sizes and resolutions. This front-end layer will interact with the business logic(server) via asynchronous JavaScript calls (AJAX) to provide a smooth and dynamic user experience (Gamma, Helm, Johnson, & Vlissides, 1994).
2. **Business Logic Layer (Server Tier):** The server. This is the heart of architecture. This layer will manage game logic, manage states, and enforce the rules of the game, handling requests from multiple clients, ensuring that only one instance of the game is in memory at any given time through session management. Additionally, it will be responsible for the game’s core algorithm and handling operations such as player management and team creation. (Silberschatz, Galvin, & Gagne, 2018).
3. **Data Storage Layer (Database Tier):** User profiles, game history, and unique identifiers will be managed in this layer. The game states need to be stored in a database, which is the purpose of this layer. The database will be designed so that there are quick reads and writes to optimize game performance. Furthermore, transactions to manage concurrency (simultaneity) and maintain data integrity across operations. (Stokes, 2003).
4. **Communication:** Communication between client and server will be handled via WebSockets or HTTP/2 for real-time gaming experience, allowing the server to push updates to the clients efficiently, minimizing latency (Coulouris, Dollimore, Kindberg, & Blair, 2011).
5. **Logical Topology:** Load balancers will distribute traffic across multiple servers (Tanenbaum & Bos, 2014).
6. **Security Measures:** Implementing robust security mat all layers, from HTTPS in the presentation layer to encryption in the data layer, addresses the critical need for protecting user data and is in line with best practices in web applications security (Stuttard & Pinto, 2011).

## [Domain Model](#_8h2ehzxfam4o)

In the UML diagram provided for “The Gaming Room”, the classes represent different components of the game application and are related as follows:

1. **Entity:** This is a parent and generic class that provides a foundation for other classes with its “id” and “name” attributes. In object-oriented programming (OOP), this is a demonstration of the principle of **inheritance**, which allows other classes to inherit properties and methods from the parent class. As a result, ensuring code reusability and diminishing redundancy.
2. **Game**: Inherits from “Entity” and represents a game session. It has an attribute “team” which is a list that can hold zero to many “Team” objects. The “Game” class has an association with the “Team” class of type composition. The relationship 0..\* indicates that a game object can have zero to many instances of a team object.
3. **Team:** Also inherits from “Entity”. It represents a team within the game. It has an attribute “players” which is a list that can contain zero to many “Player” objects. In other words, a team can exist without players or have multiple players, demonstrating a zero-to-many association relationship of type composition.
4. **Player:** This class extends “Entity” and embodies a player in the game. It stands alone in this context without further subclassing, which is a common practice for end entities in an inheritance hierarchy.
5. **Game Service:** This is a service class that manages the creation and retrieval of “**Game”** objects. It uses the singleton pattern to ensure that only one instance of “GameService” exists in the application at any given time. The presence of methods like “addGame()” and “getGame()” indicates it’s responsible for maintaining the collection of games.
6. **Program Driver:** Contains the “main()” method which is the starting point of the application. It uses “GameService”, ensuring that only one instance of “GameService” is created throughout the application’s lifecycle.
7. **SingletonTester:** This class seems to exist for testing the singleton implementation of “GameService”, ensuring that only one instance of it is created throughout the application’s lifecycle.

The UML diagram clearly follows object-oriented programming principles to establish relationships between classes. **Inheritance** is used to provide common properties to “Game”, “Team”, and “Player” classes. These classes automatically have the “id” and “name” attributes and methods defined in the parent class “Entity”. **Association** is present as well. The “Game” class has an association with the “Team” class, indicated by a line connecting them with a 0…\* notation. This notation specifies a **composition** relationship, which is form of an association implying that “Team” objects are part of “Game” objects; if a game object is destroyed, the associated “Team” objects would also cease to exist. The 0…\* means “Game” can have zero to many “Team” instances. Similarly, this same definition applies to the “Team” class, having an association with the “Player” class.

**"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** | **- Deployment method:** Mac operating systems no longer offer macOS server for hosting web services, but users can configure the built-in Apache server for local development and testing (Apple Community, 2021). MacOS is not typically used for large-scale web hosting in production environments (Rocket Yard, 2021) **- Licensing costs:** The initial investment for Mac hardware can be significant, potentially increasing the budget for The Gaming Room’s server infrastructure. | **- Deployment method:** Linux is widely recognized for its robustness and scalability in server environments, making it a preferred choice for web application hosting. Linux offers versatility and cost-effectiveness with free distributions such as Ubuntu, CentOS, and Red Hat Linux (phoenixNAP, 2021). Linux supports various web server platforms, making it suitable for nearly all web hosting requirements (Liquid Web, n.d.; phoenixNAP, n.d.) **Licensing costs:** Most Linux distributions are open-source and free, which can greatly reduce the server operating system costs for The Gaming Room. | **- Deployment method:** It can be used for hosting websites, particularly when using ASP.NET, ASP, Net Code, or SQL Server, including Internet Information Services (IIS), providing a stable and secure foundation for web hosting (IIS.net, n.d.; Kili, n.d.; WebsiteHosting.com, n.d.)​. **- Licensing costs:** The client will need to budget for Windows Server licenses, which vary based on the required capacity and specific features. | **Deployment method:** Not applicable as mobile devices function as clients, not servers. They request data from servers, which must be optimized to handle these requests efficiently. **Licensing costs:** No licensing costs associated with mobile devices on the server side. |
| **Client Side** | **- Cost:** Investment in Mac hardware and an Apple developer program membership. **- Time:** Additional time required for design and testing on Safari as well as other web browsers that run on MAC OS. **- Expertise:** Knowledge of Swift or Objective-C and familiarity with MACOS-specific development practices. **- Compatibility Requirements:** Responsive design for web apps and thorough testing MAC OS’s native browser, Safari, and others such as Firefox and Chrome. | **- Cost:** Most development tools are open-source and free, which can reduce costs.  **- Time:** Potentially more time needed for those less familiar with Linux’s command-line interface. **- Expertise:** Comfort with Linux command-line tools and a variety of programming languages. **- Compatibility Requirements:** Testing across multiple browsers and consideration for the variety of Linux distributions. | **- Cost:** Free for Visual Studio Community; fees for enterprise features or advanced editions. **- Time:** Time spent ensuring compatibility with Windows-specific browsers and features. **- Expertise:** Proficiency with .NET, C#, and Windows platform tools. **- Compatibility Requirements:** Web applications must be tested on Edge, possibly Internet Explorer, and other popular browsers on Windows. | **- Cost:** Android development tools are free; iOS development incurs costs for Mac hardware and a developer account. **- Time:** Use of cross-platform development tools can reduce time by allowing a shared codebase. **- Expertise:** Knowledge of native mobile development for both Android and iOS, and possibly cross-platform technologies like React Native. **- Compatibility Requirements:** Implementation of responsive web design for mobile browsers and adherence to mobile OS guidelines for native features. |
| **Development Tools** | **- Tools:** Xcode, Atom, Visual Studio Code  **- Impact on development team:** Requires expertise in Xcode and MacOS specific development environments. May require a dedicated team for iOS/MacOS-specific development.  **- Licensing costs:** Xcode is free, but the Apple Developer program has an annual fee. | **- Tools:** Eclipse, Sublime Text, Vim, Git, and terminal-based tools.  **- Impact on development team:** Developers need to be versatile in using command-line interfaces and various programming languages. A single team could manage due to the universal nature of web technologies.  **- Licensing costs:** Most tools are open-source and free. | **- Tools:** Visual Studio, .NET Framework, SQL Server Management Studio.  **- Impact on development team:** May require .NET specialists, possibly requiring a separate team for backend development.  **- Licensing costs:** Visual studio community is free; other editions and some Microsoft tools require payment. | **- Tools:** Android Studio, Xcode, React Native  **- Impact on development team:** Depending on whether native or cross-platform development is chosen, separate teams for Android and iOS or a unified cross-platform team may be needed.  **- Licensing costs:** Android Studio is free. Xcode requires Mac OS, which may imply hardware costs. Cross-platform tools like React Native are free, but deploying to the Apple App Store requires a paid Apple Developer account |

## Recommendations

1. **Operating Platform**: Linux is a robust operating system widely recognized for its stability and security, making it an ideal server platform for online gaming applications. It is renowned for its high scalability and performance, which is critical for handling the large user base and real-time data processing demands of this game. Linux’s open-source nature also allows for extensive customization to meet specific game requirements and ensure compatibility across various computing environments.
2. **Operating Systems Architectures**: Linux’s architecture, based on a monolithic kernel, is particularly advantageous for this game application. This architecture, where core system services like memory handling, process management, and network stack are managed in a single, large kernel space, ensuring efficient processing and quick execution of system calls, crucial for real-time gaming performance (Silberschatz, & Gagne, 2018).
3. **Storage Management**: A combination of on-premises and cloud-based solutions is advised. Cloud storage platforms like Amazon S3 or Google Cloud Storage, which are well-integrated with Linux, offer scalable, secure, and cost-effective storage solutions. These services provide redundancy and data backup capabilities, ensuring that game data is safe and recoverable in case of any system failures or data loss incidents.
4. **Memory Management**: The memory management techniques in Linux are key for gaming application’s performance. Linux uses virtual memory, which allows the use of disk space for memory, ensuring that even large games run smoothly. Additionally, the paging mechanism helps manage memory efficiently by only loading necessary parts of a program into RAM, crucial for the resource-intensive operations of this game (Silberschatz, Galvin, & Gane, 2018).
5. **Distributed Systems and Networks**: Given the requirement for the game to operate across various platforms, Linux’s networking capabilities are important. Linux supports distributed computing, meaning that the game server can communicate effectively with clients across different platforms. Implementing robust network protocols and services under Linux ensures reliable data exchange and consistent game performance. Additionally, it’s important to address potential network issues such as latency, connectivity, and handling of network outages to maintain a seamless gaming experience.
6. **Security**: Security in Linux is robust, featuring advanced user authentication, access control systems, and data encryption capabilities. These features are crucial for safeguarding user information in this game, especially in an online gaming environment where sensitive data is frequently accessed. The implementation of these security measures in Linux ensures that user data is protected against unauthorized access, a critical aspect for maintaining user trust and game integrity (Stallings, 2017).

**References**

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts (10th ed.).* Wiley

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994*). Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.

Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems (4th ed.).* Pearson

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed Systems: Concepts and Design (5th ed.)*. Addison-Wesley.

Stokes, M. (2003*). Beginning Database Design*. Wrox Press.

Wallen, J. (2010). *Linux for Dummies* (9th ed.). For Dummies.

Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook*: Finding and Exploiting Security Flaws (2nd ed.). Wiley.

Apple Community. (2021). Setting up a local web server on macOS 12 “Monterey.” Retrieved from https://discussions.apple.com

Digital.com. (n.d.). Windows Server for web hosting. Retrieved from https://digital.com/best-web-hosting/windows/

Internet Information Services (IIS). (n.d.). Home: The Official Microsoft IIS Site. Retrieved from https://www.iis.net

Kili. (n.d.). Windows Server for Web Hosting: Do You Really Need It? Retrieved from https://kili.io

Liquid Web. (n.d.). Best Server OS for Your Website: Linux or Windows? Retrieved from https://www.liquidweb.com

phoenixNAP. (2021). Server Operating System: Server OS Types & How to Choose. Retrieved from https://phoenixnap.com

Rocket Yard. (2021). Setting Up a Mac Web Server with Apache and MAMP. Retrieved from https://eshop.macsales.com

WebsiteHosting.com. (n.d.). How Windows Web Hosting Works. Retrieved from <https://websitehosting.com>

Stallings, W. (2017). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.