

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 07/16/2023 | Dre’ Scheetz | Updated Executive Summary, Requirements, Design Constraints, and Domain Model |
| 1.1 | 07/30/2023 | Dre’ Scheetz | Updated Evaluation Section |
| 1.2 | 08/20/2023 | Dre’ Scheetz | Updated Recommendation Section |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room seeks a web-based solution for their game, Draw It or Lose IT. The game is currently available on Android and is inspired by Win, Lose, or Draw. Our objective is to set up the environment and develop a real-time application that accommodates multi-player games across standard platforms. The game will render stock images as clues for teams to guess a phrase, title, or thing within the timeframe across four rounds. Our solution must include data persistence, scalability, and a user-friendly experience.

## Requirements

* A Game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player

## [Design Constraints](#_2et92p0)

* Multi-Team and Multiplayer Support: Complex data management and enhanced communication are required to support multiple teams and players, impacting development time, cost, and backend technology selection.
* Uniqueness of Names: A robust database and unique indexing mechanism are needed to ensure name uniqueness, requiring a fast and efficient look-up system.
* Singleton Game Instance: The Singleton pattern is applied to ensure a single game instance exists in memory, demanding careful memory management to prevent data corruption or loss.
* Real-Time Interaction: Real-time updates and interactions necessitate handling concurrent connections and maintaining data integrity. This impacts the choice of technologies and architecture.
* User Interface Design: Cross-platform accessibility requires responsive design, constraining front-end technology options.
* Reliability: Consistent availability under various network conditions may necessitate the implementation of failover and redundancy mechanisms.
* Scalability: The application must be scalable to accommodate user growth, necessitating a database and server architecture that can manage increased load.
* Security: Secure authentication and data protection require data encryption, secure authentication, and secure communication protocols.

## [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 below shows The Gaming Room's codebase's classes, attributes, and methods. The diagram also depicts how the classes relate to one another. Entity is a superclass or parent class that Game, Player, and Team will inherit attributes and methods from as child classes. As such, these classes extend the Entity class and can override Entity methods, known as polymorphism. Association is seen between GameService and Game, Game and Team, along with Team and Player. The relationship is shown as being zero to many. A Game may have zero or many teams, for instance. GameService is a singleton class that manages the games and ensures only one instance of the GameService object. A global point of access, getInstance(), is used to access the object while the constructor is private. ProgramDriver contains the main() method and is the system's entry point. The class initializes GameService and uses SingletonTester for validation.

Object-Oriented Programming Principles:

* Inheritance allows Game, Player, and Team classes to inherit common properties such as name and id from the Entity class, reducing code duplication.
* Polymorphism allows Game, Player, and Team to provide their implementation of the toString() method, enabling more specific and appropriate behavior for each class.
* Encapsulation allows data and states to be controlled by restricting access using private or protected fields and providing public methods.
* Abstraction allows hiding implementation details by providing a simple interface to the GameService. The complexity of managing games, teams, and players is not seen by the end user.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Characteristics: Mac servers, based on UNIX, are known for their stability and reliability.  Advantages: They offer a user-friendly interface and robust security.  Weaknesses: They can be expensive due to hardware and software costs, required to run on Apple hardware.  macOS Server was recently discontinued. macOS now offers server services. | Characteristics: Linux is open-source and flexible, allowing for various server configurations.  Advantages: Linux servers are cost-effective due to open-source licensing. They're also known for their robust performance and security.  Weaknesses: They may require specialized knowledge to set up and manage. Support might be community-based, which may only sometimes be timely. | Characteristics: Windows servers offer a user-friendly interface and wide compatibility.  Advantages: They have extensive support and integration with other Microsoft products.  Weaknesses: Licensing costs can be higher than Linux. | Mobile devices don't traditionally host servers, they communicate with servers hosted on other platforms. Even when hardware (like Android) is used for small server tasks they generally run Linux. |
| **Client Side** | Cost: Similar to Linux as most web technologies are platform-agnostic.  Time: The web app needs to be tested on all major browsers on Mac, which should not significantly add to the development time.  Expertise: Developers must be familiar with macOS and its browsers.  Smaller market share than windows for desktop; however, still relatively popular. | Cost: As Linux is open-source, the cost is generally lower.  Time: Linux supports all major browsers, so time spent on adaptation and testing won't significantly differ from other platforms.  Expertise: Developers need to be comfortable with Linux and proficient in web technologies like HTML, CSS, JavaScript, etc.  Linux has a limited share of operating system use for users. | Cost: Similar to Linux and Mac, given the platform-agnostic nature of web technologies.  Time: Testing is needed across all major browsers on Windows.  Expertise: Developers need to be comfortable with Windows and proficient in web technologies.  Windows is a popular desktop OS. | Cost: Can be more costly if a native app is needed. Similar cost to desktop if a responsive web app is designed.  Time: Could take longer due to the need for mobile-specific UI/UX considerations.  Expertise: Developers need skills in responsive design.  Both Android and iOS hold a considerable market share in the mobile space. |
| **Development Tools** | Mac developers often use JavaScript, Swift, or Objective-C, with Xcode as a popular IDE.  Some tools are free, while others require a paid license. | The main programming languages include JavaScript, Python, and PHP. IDEs could include Visual Studio Code, IntelliJ, JetBrains and other popular IDEs.  As Linux is open-source many development tools are available at no cost. | Windows supports a wide range of languages, including C#, .NET, JavaScript, Python, with IDEs like Visual Studio and JetBrains IDEs being popular.  Some tools are free, while others require a paid license. | Depending on whether it's a native or web app, tools might include Android Studio, Xcode, or Visual Studio Code. Languages might include JavaScript for web apps, Swift or Objective-C for iOS, and Java or Kotlin for Android.  Both are free to use, but deploying apps on the App Store has an associated cost. For cross-platform development, tools like React Native or Flutter can be used. |

## Recommendations

1. **Operating Platform**:

Debian, a renowned Linux distribution, is esteemed for its heightened security. While Linux is recognized as a top-tier secure operating system, Debian amplifies that security, making it apt for The Gaming Room. "Debian offers security support for its stable releases. Many other distributions and security researchers rely on Debian's security tracker." (Debian -- Reasons to Use Debian, n.d.) Debian Linux is best for extending Draw It or Lose It to diverse computing atmospheres. The hardware specifics are secondary as long as they're Linux-compatible and satisfy storage and computational demands. Notably, for those eyeing cloud storage solutions, Debian is well-represented on platforms such as Microsoft Azure.

Financial aspects and The Gaming Room's preferences might steer the decision between a dedicated on-site server or a cloud solution. "In the short term, deploying your company's on-premises infrastructure means you'll have to invest a large amount of capital upfront on hardware" (Cloud Storage Vs. On-premises Servers: 9 Things to Keep in Mind, n.d.), among other things. On-premises servers offer unparalleled control over infrastructure, while cloud solutions introduce reduced initial costs, lesser server management burdens, scalability, and immunity from physical server disasters. However, the constant need for a stable internet connection and the accumulating cloud costs must be evaluated. Without a comprehensive needs assessment, favoring one over the other is imprudent.

1. **Operating Systems Architectures**:

Debian, founded in 1993, is a beacon of stability and security in the Linux world. As a Linux distribution, it packs the Linux kernel and assorted modules tailored to diverse necessities. Linux's merits, rooted in its open-source nature and the collaborative spirit of its vast developer community, also apply to Debian. Debian offers an Amd64 cloud image architecture compatible with AWS EC2, Azure, and similar providers and backs Intel, Arm, and Amd64 hardware.

Derived from Unix, Linux boasts a monolithic kernel, signifying all OS tasks are executed at the kernel tier. Unique to Linux, its file system views all devices as files radiating from a singular root, typically granting it an edge in speed and efficiency compared to its counterparts.

1. **Storage Management**:

RAID 6 is the top pick for data storage, ensuring redundancy and protection. Furthermore, iterative backups of the codebase are vital. Both RAID 6 and backup strategies can be employed irrespective of the storage solution chosen. “Using multiple hard drives enables RAID to improve the performance of a single hard drive. Increased computer speed and reliability after a crash, depending on the configuration.” (Gillis et al., 2021).

With Linux’s file structure, separate image partitions (1.6 Gb) can be devised, distinct from the codebase. Linux supports various file systems such as EXT4, XFS, and Btrfs. For Draw It or Lose It, the EXT4 filesystem is recommended because of its stability, maturity, and journaling feature, which prevents data corruption during unexpected shutdowns. Per Salter (n.d.), “Ext4 is functionally very similar to ext3, but brings large filesystem support, improved resistance to fragmentation, higher performance, and improved timestamps.” Utilizing the hierarchical layout, user and game data can be segregated under individual directories, enhancing request handling. If leaning towards on-premises servers, storing user data on HDDs is cost-effective, while game assets can reside on faster SSDs, with backups on HDDs.

1. **Memory Management**:

Linux servers predominantly use demand paging. However, the flexibility of Linux’s memory management is commendable. Swapping, a technique to supplement RAM, can be executed by designating a swap space on a disk. “The primary function of swap space is to substitute disk space for RAM memory when real RAM fills up and more space is needed.” (Both, n.d.) Another approach involves utilizing a file within the filesystem. Regardless of the method, these serve as extensions to RAM, freeing up physical RAM for game-related operations.

1. **Distributed Systems and Networks**:

For a seamless game experience across platforms, containerization through Kubernetes is optimal. Kubernetes, an open-source gem, automates multiple facets of deployment and scaling. “Kubernetes maps out how applications should work and interact with other applications.” (*What Is Kubernetes | Ubuntu*, n.d.) A cluster of server nodes can be employed to reduce outage risks. If one node falters, others continue delivering the gameplay. Multiple hosting avenues can also be explored. For delivering images to the clients, several options are available such as a CDN (Content Delivery Network) or direct hosting. Choosing between these would need to account for The Gaming Room’s budget along with expected scale.

In addition, utilizing WebSockets is advisable. WebSockets, “an advanced technology that makes it possible to open a two-way communication session between the user’s browser and a server.” (Farhuts, 2021) This communication session is suitable for real-time game applications. Unlike HTTP, the session stays open, allowing for faster data transfer, meaning lower latency.

1. **Security**:

Debian offers integration of Security-Enhanced Linux (SELinux), a module offering advanced access controls and security policies. With SELinux, access permissions become granular, ensuring every user, from admin to players, only accesses what they should. Additionally, layers of security can be achieved with stringent authorization, robust authentication, comprehensive auditing, resilient firewalls, active intrusion detection, and steadfast end-to-end encryption. A feasible strategy is restricting remote access and implementing distinct communication channels for backend and frontend operations, ensuring secure and encrypted transmissions.

**Reference**

*About macOS Server 5.7.1 and later*. (2023, May 8). Apple Support. <https://support.apple.com/en-us/HT208312>

Both, D. (n.d.). *An introduction to swap space on Linux systems*. Opensource.com. https://opensource.com/article/18/9/swap-space-linux-systems

*Cloud storage vs. on-premises servers: 9 things to keep in mind*. (n.d.). https://www.microsoft.com/en-us/microsoft-365/business-insights-ideas/resources/cloud-storage-vs-on-premises-servers

*Debian -- Reasons to use Debian*. (n.d.). <https://www.debian.org/intro/why_debian>

Farhuts, T. (2021, December 9). WebSockets For Beginners. Part 1. - Tetiana Farhuts - Medium. *Medium*. https://medium.com/@tfarguts/websockets-for-beginners-part-1-10796106e207

Gillis, A. S., Sullivan, E., & Posey, B. (2021). RAID (redundant array of independent disks). *Storage*. https://www.techtarget.com/searchstorage/definition/RAID#:~:text=Benefits%20of%20RAID,crash%2C%20depending%20on%20the%20configuration.

*Index of /debian-cd/current/arm64/iso-cd*. (n.d.). https://cdimage.debian.org/debian-cd/current/arm64/iso-cd/

List, J. (2017, March 22). *An Android Phone Makes A Better Server Than You’d Think*. Hackaday. https://hackaday.com/2017/03/22/an-android-phone-makes-a-better-server-than-youd-think/

*Operating system market share worldwide | StatCounter Global Stats*. (n.d.). StatCounter Global Stats. <https://gs.statcounter.com/os-market-share>

Salter, J. (n.d.). *Understanding Linux filesystems: ext4 and beyond*. Opensource.com. <https://opensource.com/article/18/4/ext4-filesystem>

*SELinux Wiki*. (n.d.). https://selinuxproject.org/page/Main\_Page

Williams, M. (2021). Linux vs. Windows: How to Pick the Best Server OS for Your Website. *PCMAG*. <https://www.pcmag.com/how-to/linux-vs-windows-how-to-pick-the-best-server-os-for-your-website>

*What is Kubernetes | Ubuntu*. (n.d.). Ubuntu. https://ubuntu.com/kubernetes/what-is-kubernetes