

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 | 3/18/2025 | Phillip Wood | Initial draft of Summary, Requirements, Constraints, Architecture View, Model, Evaluation, and Recommendations. |

**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 client The Gaming Room needs their new project based on their current game Draw It or Lose It to be a web application to reach a broader audience than their current Android only app. The game rules are as follows, players compete in teams to guess puzzles progressively rendered to screen over time. The game should allow for multiple unique game instances and those instances to have multiple teams and the teams to have multiple players. The game instance, team, and player names must be unique. Each team needs to have players assigned to it. The solution to these problems will be validation logic to check for name and identifier uniqueness before adding any games, teams or players, and to

## Requirements

* Support for multiple teams.
* A team will have multiple players.
* Unique names for games, teams, and players.
* A single instance of the game in memory at any time.

## [Design Constraints](#_2et92p0)

* Single Game Instance
  + The game can have multiple games active at a time, however, those games must be unique in name, and having a unique identifier to ensure games are unique.
* Name Uniqueness
  + Game, Team, and player names must be unique. Validation logic will be required to enforce this.
* Cross-Browser Compatibility
  + The game must work seamlessly across the major web browsers. For example, Chrome, Firefox, Edge, and Safari.
* Security
  + Secure handling of the user’s data. For example, login information and game states.
* Unique Sign On
  + The user should not be allowed to be signed on and playing from multiple devices.
* Adapting From Mobile to Web-based
  + A variation in hardware, The game could run amazingly well on PC, but have performance issues on Mobile.
  + The UI must adapt to different devices. For example, desktop, tablets, mobile phones without losing functionality.
  + Network Dependency
    - Since the game is more than likely over the internet there is a possibility for lag.

## [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 object-oriented programming principles displayed in the UML are:

1. Inheritance
2. Abstraction
3. Encapsulation
4. Polymorphism

Inheritance is utilized by creating a base Entity class and deriving Game, Team, and Player cases from it. This gives the benefit of only having the id and name associated with all of the various types in one class, if a change needs to be made to all children of the Entity class it could be done in the base class. Abstraction is used by only showing the necessary details of each class instead of their implementation. Encapsulation is achieved by each class having its attributes and methods encapsulated keeping the organized and modular. Polymorphism is used with toString method in the Entity base class, so it can be overridden in derived classes to provide it’s own implementation. The GameService class shows that it is using the Singleton design pattern.

Overall, the UML sticks with OOP principles and utilizes them in way that creates a good maintainable structure for the game.

**"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 is Unix based which is known for being stable and a great environment with a lot of support for development tools. The interface is also extremely User-Friendly. However, Mac is costly, and MacOS Server is known to have limited features and compatibility issues with some tools. | Linux is Unix-like and free and open-source. It’s highly customizable to fit specific needs. Linux servers are known for stability and reliability, and great security features, and good compatibility with hosting tools allowing for a wide array of application types to be hosted. However, there are limitations with some propriety tools and applications, and the learning curve is known to be steep. Also, there is a diverse selection of distributions, which can lead to them not knowing which distribution will suit the needs of the project the best. | Windows is known to be user friendly and works great with the rest of their tech stack such as .NET and Azure, and integrates nicely with SQL Server, and Visual Studio making it a great choice for applications built with the Microsoft ecosystem. The scaling of web applications is very efficient and has great support for applications. However, it is not free and has licensing fees. Windows is more resource intensive than Linux and has been known to have security flaws in the past. | Mobile devices for the server side has many limitations, Their hardware has less processing power, storage and ram than that of a dedicated server or desktop. The operating system’s that are on mobile devices are not optimal for hosting applications. The connectivity to the internet relies on wireless technology and is slower than a wired connection. Although, there are some advantages, it can be more energy efficient than a dedicated server or desktop. They can be portable, and allow for testing from anywhere, and cost effective depending on the needs of the web-app. |
| **Client Side** | Considerations for macOS Clients are: Cross browser compatibility and testing. Responsive design principles to ensure the app adapts to screen and resolution sizes. Optimize the app for devices with newer devices using (ARM) based architecture. Minimize CPU and ram usage for smooth multitasking. Use HTTPS for secure communications and encrypt sensitive data. Use macOS’s stricter standards, such as sandboxing and permissions to user data or the file system. | Considerations for Linux Clients are: Cross browser compatibility and testing. Responsive design principles to ensure the app adapts to screen and resolution sizes. Optimize the app for lower end hardware since Linux is known to be less resource intensive older systems run it. Use techniques like lazy loading or caching. Testing across Linux environments to ensure UI/UX quality and performance. Use HTTPS, and encrypt sensitive data, also use sandboxing. | Considerations for Windows clients are: Cross browser testing to ensure the consistent behavior. Responsive design to ensure the app adapts to different screen sizes. Although windows machines can be powerful, windows can be on a tablet also. For example, the Surface, given that optimization should be to run on lower-resource devices. Implement HTTPS communication, protect against SQL injection and adhere to windows defender policies. | Considerations for mobile clients are: Use secure connections via HTTPS to ensure secure connections, Implement accessibility features, like high-contrast modes ad color blind modes. Integration with the device, so the app can send notifications. Minimize resource heavy front end operations and practice lazy loading or caching. Responsive design principles to ensure the app adapts to different resolutions and screens as well as touch gestures. Testing cross browser and multiple devices can be costly and time consuming. |
| **Development Tools** | The programming languages needed are Java for the backend, JavaScript for the frontend, and using HTML and CSS for markup and styling, and SQL for working with the database. The IDE of choice is Eclipse, a free and open-source IDE that supports web languages associated with web application development as well as Java. Utilize a Java framework for web applications such as Spring Boot. PostgreSQL and PGAdmin4 for database management. Utilize Git and GitHub for version control, and easy collaboration with pull requests and feature branches. | The programming languages needed are Java for the backend, JavaScript for the frontend, and using HTML and CSS for markup and styling, and SQL for working with the database. The IDE of choice is Eclipse, a free and open-source IDE that supports web languages associated with web application development as well as Java. Utilize a Java framework for web applications such as Spring Boot. PostgreSQL and PGAdmin4 for database management. Utilize Git and GitHub for version control, and easy collaboration with pull requests and feature branches. | The programming languages needed are Java for the backend, JavaScript for the frontend, and using HTML and CSS for markup and styling, and SQL for working with the database. The IDE of choice is Eclipse, a free and open-source IDE that supports web languages associated with web application development as well as Java. Utilize a Java framework for web applications such as Spring Boot. PostgreSQL and PGAdmin4 for database management. Utilize Git and GitHub for version control, and easy collaboration with pull requests and feature branches. | The programming languages needed are Java for the backend(A note about IOS, Java doesn’t inherently work on IOS because the JVM isn’t supported, but since the backend runs on the server and not the client Java is a fine choice.) JavaScript for the frontend, and using HTML and CSS for markup and styling, and SQL for working with the database. The IDE of choice is Eclipse, a free and open-source IDE that supports web languages associated with web application development as well as Java. Utilize a Java framework for web applications such as Spring Boot. PostgreSQL and PGAdmin4 for database management. Utilize Git and GitHub for version control, and easy collaboration with pull requests and feature branches. |

## 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**: Recommendation Linux, specifically a Debian-based distribution like Ubuntu. The package manager (APT) has most of the tools required for development in its repositories and can streamline dependency management. It is open-source and free, so it’s cost-effective. Ubuntu scales nicely and offers great security with frequent updates. The documentation is great with numerous tutorials and how-to guides, and robust reference documentation. (Ubuntu Server Documentation, 2024)
2. **Operating Systems Architectures**: The Linux kernel is monolithic, which means that all essential system services run in kernel space, including Things like device drivers and file system management. Linux offers multitasking, allowing multiple processes to execute concurrently. Virtual memory abstracts physical memory and uses paging and swapping. Linux supports multiple types of file systems, but ext4 is by far the most popular option. (GeeksForGeeks, 2023)
3. **Storage Management**: A PostgreSQL database running and being served from a cloud hosting provider such as Amazon AWS or Google Cloud Storage is the optimal solution. PostgreSQL supports replication and clustering, which is required for distributed environments. (PostgreSQL, 2020). Serving the database from a cloud service provider has multiple benefits; it’s more secure than storing data on-site and has a cheaper up-front cost. There are multiple plans associated with cloud storage, and scaling additional servers to combat higher traffic than anticipated will be much easier. (GeeksForGeeks, 2023)
4. **Memory Management**: The Linux kernel is optimized to dynamically allocate memory based on process priority, and Linux’s NUMA swap feature can be changed via commands in the CLI to boost the performance of the swaps if the system is exhausted of RAM. Linux uses Virtual memory management; this approach prevents processes from corrupting each other. Linux also utilizes buffers and caches to cache frequently used data to allow faster access (The Linux Kernel Documentation). Moreover, Java JVM uses garbage collection and will free up memory when it’s no longer used. However, garbage collection is a fantastic tool that makes programming more approachable for beginners; it’s a double-edged sword and can let the programmer make bad design choices that significantly impact the memory efficiency of the program. (GeeksForGeeks, 2016)
5. **Distributed Systems and Networks**: For the application to work seamlessly across different platforms, a distributed system can be implemented. A distributed system is essentially multiple machines running in tandem to provide a service to an end user with the illusion that it is a single computer. If one system fails, it doesn’t impact the whole system’s uptime. Utilizing a distributed system is complex, but it allows for scaling horizontally, letting multiple machines serve a single database, so the application can handle more traffic. (FreeCodeCamp, 2018)
6. **Security**: Linux provides user security using authentication features like password protection and controlled access to specific files and encryption of data. Linux is a solid choice for operating system-level security. (Tutorialspoint, 2019) However, in computer networks and distributed systems, there is more that is involved. There are numerous techniques used to increase the security and privacy of user-sensitive data; Using encryption protocols like TLS/SSL to encrypt data as it’s transported over the network, implementing or utilizing multiple-factor authentication for secure logins. Ensuring the user data is stored securely, this is another bonus of utilizing Cloud storage instead of on-site solutions. Utilize firewalls to prevent unauthorized access to servers. As well as using SSH for secure remote logins and access to servers. (GeeksForGeeks, 2022)

**References**

GeeksforGeeks. (2022, June 13). *Security in Distributed System*. GeeksforGeeks. <https://www.geeksforgeeks.org/security-in-distributed-system/>

Silberschatz, A., Galvin, P. B., & Gagne, G. (2008). *Operating system concepts*. J. Wiley & Sons.

Tutorialspoint. (2019). *Operating System - Security - Tutorialspoint*. Tutorialspoint.com. <https://www.tutorialspoint.com/operating_system/os_security.htm>

freeCodeCamp.org. (2018, April 27). *A Thorough Introduction to Distributed Systems*. <https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c/>

*Linux Memory Management Documentation — The Linux Kernel documentation*. (n.d.). Www.kernel.org. <https://www.kernel.org/doc/html/v4.18/vm/index.html>

GeeksforGeeks. (2016, February 24). *Garbage Collection in Java*. GeeksforGeeks. <https://www.geeksforgeeks.org/garbage-collection-java>

PostgreSQL. (2020, November 12). *PostgreSQL 13.1 Documentation*. PostgreSQL Documentation. <https://www.postgresql.org/docs/current/>

GeeksForGeeks. (2023, January 7). *10 Advantages and Disadvantages of Cloud Storage*. GeeksforGeeks. <https://www.geeksforgeeks.org/10-advantages-and-disadvantages-of-cloud-storage/>

GeeksforGeeks. (2023, September 25). *Architecture of linux operating system*. GeeksforGeeks. <https://www.geeksforgeeks.org/architecture-of-linux-operating-system/>

*Ubuntu Server documentation*. (2024). Ubuntu Server.

<https://documentation.ubuntu.com/server/>