

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc185199459)

[Table of Contents 2](#_Toc185199460)

[Document Revision History 2](#_Toc185199461)

[Executive Summary 3](#_Toc185199462)

[Design Constraints 3](#_Toc185199463)

[System Architecture View 3](#_Toc185199464)

[Domain Model 3](#_Toc185199465)

[Evaluation 4](#_Toc185199466)

[Recommendations 7](#_Toc185199467)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <11/16/2024> | <Johnathan Smith> | Original creation of Design |
| 1.1 | <11/29/2024> | <Johnathan Smith> | Edit of the table under Evaluation |
| 1.2 | <12/13/2024> | <Johnathan Smith> | Edit to 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 Gaming Room has an Android-only application called ‘Draw It or Lose It’ that they would like to develop as a web-based game that works on multiple platforms. In order for the application to function properly, each game, team and player must be unique. The program uses design patterns such as Singleton and Iterator, that will check the games and access lists for games, teams and players to ensure that no duplicate instances are created, and everything is unique. This staff needs help setting up the environment.

## [Design Constraints](#_2et92p0)

The Gaming Room’s current application is Android-only and the API will need to be extended to work with the new web-based environment. The application will have to be cross-platform, requiring the use of server-client architecture. All Identifiers in the game must be unique to ensure that everything is limited to a single instance.

## [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)

Entity: The Entity base class holds the common attributes of id and name for all the extended classes (game, team, and player) within the application.

ProgramDriver: The ProgramDriver class holds the main method of the application, including the application outputs and the use of the singleton tester.

GameService: The GameService class initiates the single instance of the service, while blocking any additional instances of the service occurring. This class holds the game list and is used to add a new game to the list.

SingletonTester: This class is a singleton tester used to ensure that there is only one instance of GameService class is running.

Game: The Game class inherits from the Entity parent class and holds the team list and methods to add teams to the game, while using an iterator to search through the teams list, ensuring that each team has a unique id and name.

Team: The Team class inherits from the Entity parent class and holds the player list and methods to add players to the game, while using an iterator to search through the players list, ensuring that each player has a unique id and name.

Player: The Player class inherits from the Entity parent class and holds a unique id and name.

Polymorphism: The toString method is overrode within the classes to show a specific string for each object.

Inheritance: The game, team, and player classes inherit from the base class ‘Entity’ to share the id and name methods.

Encapsulation: The data for the game, team, and player entities are encapsulated within their respective class, so as to avoid modification and access.

**"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** | Using Mac for a server is a more expensive route and is limited on support in comparison to Linux. It integrates the best if used with other Apple solutions. The server costs for Mac are low, but when compared with the price of the required hardware needed, things can get expensive quickly. | Linux is low cost and open source. It is free, with a large support for web servers. It has a big amount of distribution options. There is a slight learning curve for new users of Linux, but the result is a customized experience that can be tailored to specific needs. | Windows servers have great support for applications, with a user-friendly interface and advanced security. Windows servers can be costly and may not perform well in some scenarios. Windows is lacking in mobile development options, so making changes on the go can be difficult. | Mobile devices lack the needed power to be useful for hosting a server, therefore it is rarely used for such. Mobile devices are convenient for on-the-go development and testing. Mobile devices lack advanced security options, making them more vulnerable for server-side development. |
| **Client Side** | The cost of the hardware required for Mac is expensive and would require someone with expertise in Mac SDKs. Achieving cross-platform support would take longer on Mac. Testing on multiple web browsers would be required for ensuring the application is compatible. | Being open source, using Linux will have a cost on the lower side. This will require someone familiar with the Linux environment, though the larger community means a bigger applicant pool in comparison to Mac. Multiple testing methods must be used to ensure correct functionality across the board. | Windows has a large user base, with a moderate cost in comparison to Mac. Turn-round times are good with Windows, with its ease of use and big support system. Someone with expertise in the .NET framework would be required. Windows may be less customizable than Linux is some scenarios. | Required developers would need expertise in the development of mobile applications. Knowledge of development across both iOS (Swift) and Android (Java) is necessary. This can lead to longer development times. The mobile device scene has a very large user base. |
| **Development Tools** | Mac uses Swift and the Xcode IDE for its development solutions. It is limited to Apple solutions. This limitation means the development team must be skilled with Apples tools, increasing the potential development cost of a cross-platform application. | Linux has access to many programming languages and tools, such as C, C++ and Python, as well as IDE tools like IntelliJ and Eclipse. A development team could work across multiple languages and tools, widening the possibilities involving both cost and outcome of the application. Linux is mostly free open-source material, so it has more financial resources to hire a highly skilled team. | Visual Studio and .NET framework, with the most widely used C# would build this type of software for Windows. A development team trained in the .NET framework would be essential, raising the cost of a strong team. The use of the .NET framework is free, which will help offset the costs of a specialized development team. | In this scenario, Swift and Xcode would be used for development on iOS, with Android using Java and the Android Studio IDE. The development team for mobile device implementation will require skills in both Android and iOS, so developers will either need experience on both platforms, or two individual teams must be hired for this development. |

## 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:

I suggest using Linux as the platform for the ‘Draw It or Lose It’ application. Ubuntu Server LTS would be an appropriate choice because of its stable operation, great community support, and periodic updates. The game can be easily scaled up/down in response to demand with a strong environment. It's also cross-compatible with a variety of hardware, which means it can be used under a wide spectrum of server architectures and provides a flexible solution.

2. Operating Systems Architectures:

Ubuntu Server is suggested to be used, as it is based on a single monolithic Linux kernel which is reliable and optimized, providing the primary functionality of the system, such as process management, memory management, and device drivers, in a single giant process. This architecture reduces the number of context switches, and the overhead due to this inter-process communication is remarkably low. For this, it would be suitable to use Apache or Nginx among the available web servers. Apache provides fine control of its configuration and a multitude of modules to extend its behavior, but Nginx provides the required high performance with minimal resource consumption. For handling several thousands of concurrent connections, it should be fine.

3. Storage Management:

In storage management, MySQL is the database management system. MySQL is an open-source relational database management system whose main responsibilities are handling complex queries, transactions, and data integrity. This could be used for data storage including games, teams, and players. In-memory caching can be achieved using Memcached to speed up and improve responsiveness. Memcached caches frequently accessed data in RAM, reducing the load imposed upon the database and enabling fast access to database data for real-time gaming interactions.

4. Memory Management:

Linux efficiently handles memory by using the virtual memory of its architecture that joins physical memory with disk memory to create an address space that is larger than the capacity of the physical memory. This ability enables the system to process more data and more applications. In ‘Draw It or Lose It’, the images and data used for the current game must be in memory, then these images and data must be transferred to RAM to ensure consistent frame rates. Paging and swapping could move the inactive data out to disk, freeing the RAM for other, more important, uses. Correct caching size and proactive control of memory allocation are of utmost importance for performance.

5. Distributed Systems and Networks:

WebSockets can be deployed to implement real-time and bidirectional communication between the server and clients to provide seamless communications for multiple platforms. It provides low-latency, persistent connections required for ‘Draw It or Lose It’. Non-real-time communications (managing game sessions, getting game histories) will be handled through the RESTful APIs. These APIs will enable smooth communication among the different components of the application, regardless of the platform used. Handling outages with fallbacks should be provided so continuous gaming operation can be sustained.

6. Security:

A firewall, like UFW, will prevent unauthorized access and intrusion into the network. Intrusion detection systems (Suricata) can be instantiated and operated on network traffic to identify behavior and potential threats. SSH should be implemented with key-based authentication for user authentication in place of passwords to mitigate the vulnerability of brute-force attacks. HTTPS with SSL/TLS encryption will guarantee that all communication data moving from the client to the server is encrypted and protected from manipulation.