

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 | 10/20/2024 | James D. Webb | This document aims to consider the different benefits and drawbacks of choosing a specific operating system to host a web server, while also taking into account the various software limitations of each. |

**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 wants a game where teams compete against each other by guessing what is being rendered in an image. There needs to be at least two teams with multiple members on each team. The staff at The Gaming Room do not know how to implement the environment for such a game and need help.

## Requirements

The requirements are to have two teams with multiple players assigned to each team. Names must be unique, and the system should check if a name is already in use. Additionally, only one instance of the game can exist in memory at a time

## [Design Constraints](#_2et92p0)

The most difficult part of design constraints is having the software run on multiple operating systems (OS) while choosing which OS to host the application on. This includes various barriers such as cost, time and compatibility.

## [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 Entity class is the main or base class for Game, Team, and Player. It provides properties such as id for the players and name. It also has various methods, such as getId() to get the ID and getName() to get the name. The Game class inherits all the properties from the Entity class, including those in the Team and Player classes. The Game class also indicates that it has zero or more teams. The Team class indicates that it can have multiple players, providing a list of players as attributes and various methods for adding a player to the team. The Player class holds basic details such as name and ID. These three classes demonstrate object-oriented principles in how they interact with each other, including inheritance, encapsulation, and composition.

**"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)

After learning the various software application requirements, I can see that Java and various frameworks are going to be essential in building cross platform applications. Linux and Mac are more Unix based applications that can be developed with python, Java and Ruby on rails. The same with Linux is Java python and the native web servers such as Apache. Windows has its own set of applications such as c# and .net, however there are IDE and programming languages to compensate for this. Mobile devices are mostly Java within their framework, including python and various other standard http and html functions.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS is a versatile operating system with feasible applications for developing software for both Apple devices and cross-platform environments. macOS utilizes built-in programming languages such as Java and Python, along with its Unix-based system. This provides good versatility for programming and running applications across different operating systems. | Linux, similar to macOS, is a Unix-based system that natively supports Java, Python, and various other programming languages. This allows applications to be developed and run on Linux using a variety of languages. Both macOS and Linux support Node.js and Python frameworks, which are efficient for running on servers. | Windows is a highly versatile operating system, offering support not only through software applications but also in terms of functionality. Being the most widely used operating system, Windows supports various programming languages and IDEs for developing applications. | Mobile devices are limited in what they can do but are specific when it comes to programming applications. Applications must be tailored to various operating system versions and hardware requirements to support different screen sizes and features. |
| **Client Side** | macOS supports various languages such as Swift, C, JavaScript, and Python natively. This allows applications to be developed easily for various macOS-supported frameworks and APIs, including WebSocket protocols and security features such as SSL/TLS. | Linux, similar to macOS, supports Node.js, Python, and Java natively, which allows various applications to be created and maintained on a server quite easily. It also supports security protocols such as SSL/TLS and WebSockets. | Windows, being the most widely used OS, supports a wide range of features, including driver compatibility, software, and peripheral compatibility. This allows Windows to support almost any native server as a client. | Mobile devices primarily work with Java-based APIs, which allow for the easy development of applications for these devices. The challenging part is ensuring compatibility with the wide variety of screen sizes and hardware components. Additionally, there are various cross-platform APIs available for developing applications on mobile devices. |
| **Development Tools** | Mac offers a wide variety of IDEs for building native and cross-platform applications. Xcode, being the most popular native software IDE, is used for developing applications for all Apple devices. It supports various languages such as Swift, C++, and Objective-C. Mac also supports Microsoft’s Visual Studio, a lightweight IDE that, through plug-ins, can support various programming languages. | Linux, being open source, offers a wide range of IDEs for application development. Visual Studio Code, being the most prominent, allows for native and cross-platform development with extensive libraries. Linux also supports Eclipse, which is more suited for Java programming, and PyCharm, which is specific for Python. | Windows offers Visual Studio, which is best suited for developing native applications and cross-platform projects utilizing a wide range of libraries and packages. Windows can also use the lightweight version, Visual Studio Code. Similar to Linux, Windows supports other programming language-specific IDEs such as Eclipse and PyCharm. | Depending on whether the device is Android or Apple, both support native IDEs such as Android Studio and Xcode. Mobile devices also support Visual Studio and Visual Studio Code, which offer a wide range of libraries and extensions. React Native is also a popular choice for building cross-platform applications in JavaScript, which mobile devices are well-suited for. |

## 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**: With a majority of websites being hosted on Linux-based servers, I would recommend Linux for hosting The Gaming Room’s web-based service. While Linux requires more of a learning curve, it has become more popular and will be suitable in terms of finding expert personnel to manage the server. Linux is also the most cost-effective choice in terms of hardware and software and will allow scalability with ease.
2. **Operating Systems Architectures**: Linux offers a variety of operating systems, including Red Hat Enterprise Linux, Ubuntu Server, Debian, and CentOS. It is compatible with most hardware, and being open-source allows for abundant support in terms of compatibility. Linux is also lightweight and configurable, offering speed and performance. Additionally, Linux provides open-source and secure web servers such as Apache and Nginx.
3. **Storage Management**: Linux is compatible with various RAID configurations depending on scalability and needs. It supports both hardware and software-specific RAID controllers. For example, Linux can support RAID 1 for improved read and write speeds if there is a large number of clients, although the storage capacity for RAID 1 is cut in half. However, RAID 1 is limited in failover configurations. RAID 10, for instance, allows for speed while also maintaining failover features such as mirroring, so data is not lost.
4. **Memory Management**: The biggest benefit of memory management in Linux is that it is open-source and has a large community providing continuous support for improvement and troubleshooting. Being open-source allows for various techniques and tuning within the kernel that are not available for proprietary OSs. Linux also supports security features such as memory protection and kernel same page merging.
5. **Distributed Systems and Networks**: While Linux supports a plethora of middleware that enables various protocols and communication between platforms and APIs, it also natively supports protocols such as HTTP/HTTPS, TCP/IP, and SSH. Linux can also utilize various security encryption methods and provide fault tolerance for networks. Additionally, Linux supports various monitoring and logging tools, such as Prometheus and Nagios, which can provide insights into network performance and system status, including MySQL and Azure SaaS monitoring.
6. **Security**: While it takes technical skills to manage the security of Linux, it is renowned for its security features by design. Linux requires administrators to be more proactive in applying security patches. However, the result is a more robust security environment, including a centralized authentication system, strong authentication requirements, and full disk encryption. As mentioned before, Linux requires strong technical knowledge to maintain its security posture at the highest level. However, if implemented correctly, Linux can be the most robust and secure OS for hosting a web application server.