

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 | <03/20/2025> | Christopher Diaz | I will be implementing the Entity class into its own file and using it to simplify the structure of the program and also to enhance the functionality. The entity class will extend other classes to assure that player, team, and game name remain unique, with no duplciates. |

## [Executive Summary](#_sbfa50wo7nsh)

The client wants to implement a game application that manages only a single instance of the game at once, so no other instances should be saved in the memory. For each instance of the game run, there has to be unique names for the game, the teams, and the players, so there can be no duplicates. To achieve this, I can use a singleton pattern which will make sure only 1 instance exists in the memory, and I can use other methods like inheritance and iteration. A combination of these methods will ensure that the game functions as intended while also maintaining clean and organized code.

## [Design Constraints](#_2et92p0)

In terms of constraints for this application, the biggest one is the scaling of it. We must consider the handling of a large number of teams and players while also considering performance at the same time. Aside from the scaling of it, another important factor is the security. We must be able to keep user data secure and protected away from any unauthorized users. The user experience is another important constraint to keep in mind of at all times. The application must be in a state where it can be updated to enhance user experience and make it even better.

## [Domain Model](#_8h2ehzxfam4o)

Among these classes, the Game, Team, and Player class will inherit from the Entity class, which involves common attributes like the id and name. Each game will contain a list of Team Objects and each team will contain a list of Player objects. GameService will be implemented as a singleton class, which will ensure that only 1 instance of GameService exists at any time. Additionally, ProgramDriver will use the SingletonTester class to test the singleton behavior which will make sure that the app functions properly. Among these classes, different methods such as encapsulation, inheritance, abstraction, and use of the singleton pattern are used in order to obtain and achieve the desired outcome and results of the application, as requested by the 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)

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** | Unix-based OS with limited server capabilities. It can handle and support developer-friendly programs like python and Node.js, and it is highly secure. While it has its benefits, there are downsides like how it is not best designed for server hosting. There is also cost to consider, and running on its hardware and licensing can be very expensive. Scaling with Apple hardware can be costly as well | An open-source Unix-like OS that dominates in web hosting. It is extremely customizable, making it capable of working for different types of server workloads. It is also extremely stable and can remain that way for a very long time if optimized properly. With Linux, the cost for licensing is low to non-existent, and the hardware requirement is light, so it does not need expensive hardware to run efficiently. There are frequent security updates to keep it secure, it has a wide selection of software that can be used for development, and it is easily scalable and can easily add more servers. The biggest downside could be the lack of enterprise support, and that it has a decently high learning curve to really be good at as a developer. | A server OS developed by Microsoft with integration between Microsoft services and software. The Windows interface is also something all developers would be familiar with, so the learning curve is smaller, and easier to grasp. There are constant updates and excellent support, making it a secure option that contains a plethora of software that is able to handle and withstand a variety of different development purposes. On the downside, it can be costly, and heavily needy in terms of hardware. There can also be some issues with scalability | Mobile OS involves Android and IOS which are both limited in several areas such as their capability to handle server workloads. The main benefits are its portability, and its low power consumption, making it useful for more unique situations where the other platforms may not be available in. On the downside, because of its light hardware specs, it is limited in areas such as the performance and scalability. Additionally, it can pose a security risk, and it requires the developers to adhere to Appstore requirements and restrictions. |
| **Client Side** | Requires frontend developers, UI/UX designers, and QA specialists for testing. The overall time it could take can be up to just around 3 months. When it comes to cost, it will widely vary, but it can cost anywhere between $20000 to $50000 | Requires frontend developers, system specialists, and QA specialists/engineers. The time it takes to be ready for deployment can be anywhere between 3 to 4, or even 5 months. In terms of cost, depending on factors like the time it will take, it can cost anywhere between $10000 and $35000 | Requires frontend developers, devOps engineers, and QA specialists/engineers. The time it takes to be ready for deployment can be anywhere between 1 to 3 months. As for the cost, it will be dependent on several factors that can make it cost up to $19000 to $38000 | Will require mobile developers, UI designers, and QA specialists/engineers. The deployment can take anywhere between 3 to 6 months, making it the longest due to needs for testing and functionality across all mobile platforms. The cost can be anywhere between $35000 to $75000 |
| **Development Tools** | Xcode, VS code, or WebStorm could be viable tools used by the developers. Using these tools, the developers must implement JavaScript, HTML, CSS, and even Swift. | VS code and Eclipse alongside a browser tool like chromium or Firefox will be dominantly used. This is to utilize programming languages and frameworks like JavaScript, HTML, CSS, React, Vue, Node.js, Python, and other similar programming languages which will allow deployment to be successful on Linux | VS code, Visual Studio, Python, Eclipse, and WebStorm can be used alongside a browser tool like Edge DevTools or WebView2. This will allow the developers to utilize programming languages and frameworks like HTML, CSS, JavaScript, React, Angular, .NET core, Node.js, and SQL server | Xcode, Android Studio, and VS code will be viable tools for the developers. They will be able to implement HTML, CSS, JavaScript, React, Python, Node.js, Swift, and IOS and Android simulators/emulators for testing. Additionally, WebKit will be needed for IOS while chromium will be needed for Android |

## 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**:

A suitable operating server platform would be a setup like a Linux-based server deploying a cloud platform like Amazon Web Services or Google Cloud Platform. Linux is strongly suitable for a mix of web applications, mobile platforms like Android, and APIs because there is high compatibility with relevant languages and frameworks like Node.js and Python which are essential for allowing this cross-functionality. Aside from how well everything works together, Linux has a cheap cost to build and maintain because a lot of what is necessary like Ubuntu is free and open source. There is also the benefit of it being highly scalable since Linux is lightweight and can be optimized for performance.

1. **Operating Systems Architectures**:

Linux has a monolithic kernel, so it contains parts like the file systems and networking in a single kernel space which is meant to enhance speeds. It is also useful for supporting multiple users and processes happening at once, giving ease in multitasking. This will overall improve performance efficiency because of low-latency calls and vertical/horizontal scaling

**Storage Management**:

For storage, using a combination of a cloud-based storage mixed with a database like PostgreSQL will be the best route. The database can handle instances of the games, team and player history, scores, and session data. Cloud-based storage can be used for storing the images so that the program itself does not have to be hindered by having a massive load of images to process and load.

1. **Memory Management**:

Using Linux, each game can have its own virtual memory space which will ultimately help in preventing crashes or bugs that happen in one instance from happening in other instances that are running at the same time. Linux will also use the memory that is available to improve and enhance performance, and then it can automatically reclaim that memory when needed, like when a round ends. Additionally, using a framework like Node.js or software like Python will allow memory management to be efficient and clean since they can handle garbage collection. This will ensure that regardless of how long the game is, performance remains high and optimized.

1. **Distributed Systems and Networks**:

For communication, utilization of something like web sockets will allow for live communication between the server and clients, which will be useful for updating the images, guesses, and timers. Alongside web sockets, REST APIs will be useful for operations like creating the game, joining a team, and submitting an answer. To make the backend scalable, utilization of software like Docker will allow the utilization of containers for distributed, scalable services.

1. **Security**:

When it comes to security, we have to make sure that only authorized users are allowed to log in and join teams while also making sure that these users are not permitted to do anything more than what is allowed, like guessing the image. Encrypting data with something like HTTPS for communication between browsers and servers will make sure that not just anyone can go in and read and tamper with it. The addition of a firewall and network protection will keep any suspicious traffic and activity under control by protecting from attacks and unwanted attempts to sign in and gain access to information like user data. To make it even more secure, using secure APIs can protect the backend from unauthorized access as well. This will make the application require API tokens or IDs for game-related requests. To stack on to all of this, adding constant updates and patches to keep the application up to date will be essential for its security, and finally, utilizing tools like Fail2ban or AWS shield will go hand in hand in making sure that the application is secure from any new threats that may rise because it will always be monitored and it will always have new and better security with the more patches that are implemented.