

<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 | <09/17/23> | <Ugochuku Enyi Ebere> | Client website to mobile |
| 1.1 | <09/30/23> | <Ugochuku Enyi Ebere> | Operating System evaluations |
| 1.2 | <10/13/23> | <Ugochuku Enyi Ebere> | Operating System Recommendation |

**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)

<*Chat Away, a successful social media company, has a website. They intend to push the website/service to more users through a mobile application version of the website. This will allow them to reach more users and bring in more revenue*.>

## Requirements

*<* *Create an app that appeals to a wider audience through a mobile app which in turn, increases revenue. The app must work on both iOS and Android platforms, and stay within their price range/budget.>*

## [Design Constraints](#_2et92p0)

<*1. The budget ceiling is recommended to be adhered to.*

*2. Software should be available on IOS and Android, app appearance and UI might differ*

*3. The project timeline could be weird since we are making a website into an app to fit various platforms*.>

## [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)

<In the diagram we have the driver class containing main. This will allow us to begin to create objects of other classes(game, team, player).

We then have GameService. It employs the singleton pattern which restricts the instantiation of the class to a singular instance in memory. We can instantiate with the getInstance(): GameService method.

The addGame() method creates a new game object and adds it to the list of games. From there, we can create a team with the use of the addTeam() method, whereby the team is then added to a list of teams. The same applies to player. A player object is created with the addPlayer(name): player method. These classes use the iterator pattern to prevent objects of similar names and types from being added to the lists.

Game, Team, and Player inherit attributes from Entity. The whole structure makes use of things like inheritance and polymorphism, abstraction, and encapsulation as well.>

**"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** | <Macs seem to be able to host a bunch of web-based applications, with the environment, functionality, and user UI being among the most sought-after for applications centered on editing. It can host game applications, but not as well as other platforms. Users would most likely seek out other platforms for gaming..> | <Linux is quite popular. It is a stable and reliable platform very capable of hosting such applications. It is free, easy to maintain, and runs on any hardware, especially older hardware. One other thing to note is its great security as it is less prone to malware.> | <Most, if not all computers come with Windows operating software. Due to this, just about anyone in any home can access an application Windows supports. It is highly capable of hosting web-based applications, is easy to use, and is endorsed by many hardware vendors. However, Windows isn’t free and is more prone to malware than most operating systems, due in part to its great ability to support third-party software.> | <Mobile devices are more commonplace than any operating system being considered right now. They contain so many features and inbuilt capabilities. They are easy to find and use and are portable. They are capable of hosting web-based applications. However, due to the different features of different mobile devices, such as screen size, and resolutions, creating an app that works well on mobile can be challenging, this is also contributed to by a lack of resources, or rather, a streamlining of resources.> |
| **Client Side** | <People who buy Macs usually use them for business, production, and creation. To reach such a market, things like aesthetics and simplicity must be considered. In other words, the game has to run smoothly and look neat, organized, and polished for it to be endorsed on the platform.> | <Due to how Linux works on older hardware and its common characteristics such as high customization and high security, intensive testing will be required, which will cost both time and expertise. Since it is open source, things like cost can be managed.> | <Windows, while near perfect for all things being considered, is not free. Licensing costs would be involved. However, due to its great compatibility with almost every software being created, it could prove useful.> | <Mobile devices cannot be considered as one single platform like the rest, as things like screen size and resolution come into play. It would take a significant amount of time to develop and test the application in all environments and screen sizes.> |
| **Development Tools** | <JavaScript is the fundamental language for building any web application, this way it runs on most, if not all web browsers. An IDE that can run JavaScript would be Visual Studio. It allows for development on other specific platforms and languages as well.> | <Java, python, c++, any of these languages can be used, but Java takes the cake as many devices contain Java virtual machines. Visual Studio and Eclipse for Java could serve as the IDEs.> | <Java, Python, and C languages, can help us build this application on Windows, and as such, some of the most popular IDEs such as Eclipse for Java, pycharm for Python, and Visual Studio for all three but mainly C will help..> | <Java has been the primary language for Android app development, which coupled with Swift for iOS will help us create the application. The IDEs used will be Android Studio and Xcode or Flutter for iOS.> |

## 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**: <From all operating systems considered, I would recommend Windows or Linux. Windows has a massive, probably the biggest amount of users. Almost all computers that are distributed today run on Windows. Due to this, it can be used almost seamlessly with all mobile platforms. Linux on the other hand is stable and reliable, as mentioned before, it is less prone to crashes and viruses. However, for one reason, being the Ubuntu server, Linux takes the cake. A thing I was worried about was community support as Windows seemed the most appealing due to its large user base, but Linux has a good one as well.>
2. **Operating Systems Architectures**: <Linux is composed of several components – the kernel(core component) providing services for hardware and acting as a bridge between it(hardware and software), as well as managing system resources like the CPU and memory. We have the shell, which interprets user commands and executes programs, we have the file system implemented by the kernel, and then we have device drivers, which allow the OS to communicate with hardware. Linux has a lot of open-source drivers which could help provide support for various hardware.>
3. **Storage Management**: <Cloud storage seems to be the way to go. It is easily adjustable at a fixed price, and more than one game service can be started up/instantiated. Cloud service also provides durable and highly available storage options, with no need to worry about storage on a local device.>
4. **Memory Management**: <Linux uses virtual memory, allowing the draw it or lose it application to use more memory than might be available through a technique called swapping. It involves moving data from the RAM when full to free up RAM for active processes, making sure that there is available memory when needed.>
5. **Distributed Systems and Networks**: <Cloud service providers for the win once more. We distribute cloud services anywhere, meaning multiple servers and locations, helping us curb outages. We could also make a client that in the event of an outage, it can store information that would notify the server when it comes back online.>
6. **Security**: <we could simply use role-based authentication and passwords. Assigning the minimum necessary permissions for users to perform tasks on the game to minimize the chances of a security/data breach. Authentication policies such as passwords, or two to multifactor authentication.>