

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/16/2022 | Brian Bentley | Original release and development. |
| 1.1 | 09/30/2022 | Brian Bentley | Added section to evaluation. |
| 1.2 | 10/02/2022 | Brian Bentley | Made additional clarifications in the evaluation section to reflect the client/server side relationship. |
| 1.3 | 10/10/2022 | Brian Bentley | Finished the section regarding our recommendation for this software. |

**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)

As a consultant for Creative Technology Solutions (CTS), we are working with a new client called, The Gaming Room. The Gaming Room is looking to develop a web-based game that will cross over into multiple platforms based off their current game, Draw It or Lost It. This app is currently available only on Android. This game will have teams compete to guess what is being drawn. The app will choose from a library of stock images as the clues. The game consists of four rounds, which last one minute each. The drawings are rendered steadily until they complete at the 30-second mark. If the current team does not guess the puzzle before the times up the opposing team will have the opportunity to guess with an additional 15 seconds.

## [Design Constraints](#_2et92p0)

* Game and Team names will have to be unique to allow players to check for a unique name. There can only be one instance of the Game at a time.
* This game must run on multiple platforms.
* The game can be played with one or more teams, there will be multiple teammates.
* We will have to make sure to be ready to code in other languages such as Swift and C++ since we are looking to make this a multi-platform project.
* We can also investigate using the code in the existing Android application and using inheritance from other platform specific languages.

## [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 superclass will inherit the Game, Team, and Player classes. This means that Game, Player, and Team are all Entities. The Game service subclass will be related to the Game class. Also, you can see that a Gameservice and have multiple games, and a game can have multiple teams and so on. Game(s) will have a team, and team(s) with have a player(s). In the upper lefthand corner you can see that the ProgramDriver will run the main function, this is used by the SingletonTester to test for a Singleton.

**"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** | Utilizes a terminal like Linux to configure and setup servers.  Mac is not usually the preferred service for web hosting, and thus might not be as easily used as the other services. Great tech support, however. Mac does not ship with J2EE implementation however you can set it up for a Mac Server side application. Since MacOS servers will not longer be a thing from 2022 on it doesn’t make sense to implement this. | Linux is known for using a terminal to quickly and easily configure servers. Linux also uses cost friendly distros built on being used as web hosts. Linux is also known for being a secure service since it is open source. For HTTP based applications you can apt-get the Node.js webserver which would implement the http server-side architecture that we are looking for. This can handle authentication and authorization. | Windows has wide usage compared to the other platforms. It is known for its “comfortability”. Windows is not known for being the most secure OS. Microsoft can have terrible tech support. Can use WinHTTP to support desktop client applications that can handle authorization and authentication for multiple clients when written correctly. | Servers should be hosted in static locations as opposed to being mobile.  If hosting a server on a mobile device, you can expect there to be little upgradeability or real server infrastructure. Good for micro traffic and mobility. Good to use Django for server-side applications |
| **Client Side** | Mac is a UNIX based system and thus inherits some features from that OS.  It is very user friendly, even to an annoying degree.  Very secure OS architecture.  Will be a moderately difficult task to develop on MAC if not used to their IDE’s. Apple apparently has HTTP API’s that are included in the OS. Supposedly Xcode. | Linux will be the most difficult OS to work with in regard to expertise and time required. The cost will be minimal however since it is open source. In the end the customizability of Linux to make it support multiple clients can end up being easier than the other systems since you often have to build it from the ground up over time.  Javascripts framework and Node.JS would be able to serve for the client-side development for our application and includes many ways to communicate with the server. | It is easier to find experts willing to build and develop with windows-based systems, but the costs will be just as much if not more than Mac due to their mindset of proprietary software. Although Windows has been getting more open source lately. Easily the most software to complete the task at hand. Javascripts framework and Node.JS would be able to serve for the client-side development for our application and includes many ways to communicate with the server. | Will be more flexible and provide more mobility to clients and developers. Probably more difficult to implement than other OS and devices. Will require experts in mobile development languages. Javascripts framework and Node.JS would be able to serve for the client-side development for our application and includes many ways to communicate with the server. |
| **Development Tools** | Swift is by far the most popular option for developing on MACs. MACs can still run many other languages. It can use HTML/CSS/JavaScript, Java, Python, Ruby, and so on. MAC can use many of the popular IDE’s available online. Apple HTTP API’s. | Linux works with eclipse and visual studio. There is a plethora of languages that Linux can use to get the job done. You can also use bash scripting for setting up servers. Can develop with nearly every language if downloading the right libraries and distros. JavaScript, Node.js | Will work with as many languages as Linux but will have an easier time doing so since most software development tools are developed for Windows machines. Can use Python, Java, C++, PHP, Ruby. Can also utilize the PowerShell for scripting. Django and WinHTTP are tools to use for http-based applications. Can also develop with Node.js/JavaScript for http-based applications. | Android and Swift are the dominant languages for mobile development. These languages can be run on any of the other operating systems. There are now options to also develop with the typical big languages such as C++, Java, and Python even if the libraries are not as fully fleshed out as Android and Swift. JavaScript, Node.js. |

Now that we can see what the differences are here, lets further break down the Server Side, Client Side, and Development tools.

**Server Side –**

Windows and Linux have server-based deployments. MacOS however has been discontinued! The potential licensing costs can range from nothing (using Linux) to a very large portion of a company’s budget if using Windows. Linux servers can be very fast and resource friendly when utilized correctly, as well as being easy to scale out quite quickly. Windows has this same ability, but with less customization for the applications you are using.

Windows:

Proprietary

Secure

Expensive

User Support

WinHTTP

Can utilize a large amount of software to perform client-server interactions

Linux:

Open Source

Free

Secure

No support, must use forums or find Linux experts

Can apt-get packages to utilize the client-server interactions

**Client Side-**

With these different OS there are varying costs that will come with developing an application on multiple types of clients.

Windows:

Cost – Expensive

Expertise – Least amount of expertise

Time – Least amount of time

Linux:

Cost – If not free, then least expensive

Expertise – Most amount of expertise

Time – Most amount of time required

Apple:

Cost – Expensive

Expertise – Moderate amount of expertise

Time – Moderate amount of time.

Mobile:

Cost – Moderate

Expertise – Relatively moderate

Time – Moderate

As we can see, even though Windows is the most expensive to develop applications on it is also the easiest to develop applications on. The other OS’s have varying degrees of time and difficulty and depending on turn around time, that might be prohibitive to the project and its investors.

**Development Tools –**

Windows –

* High licensing costs
* Development difficulty will be minimal, will need only a few developers.

Tools

1. Visual Studio
2. Eclipse
3. Pycharm
4. Gitlab
5. Node.Js

Languages

1. C++
2. Java
3. Python
4. JavaScript

Mac –

* High licensing costs
* Development will be moderately difficult will likely need a medium sized team of developers.

Tools

1. Flexihub
2. Homebrew
3. Xcode

Languages

1. Swift

Linux –

* No licensing costs
* Development is the most difficult, will have to have a decently sized team of developers and linux tech support to build from the ground up

Tools

1. Eclipse
2. Node.js
3. VIM
4. Gedit
5. Pycharm
6. Atom

Languages

1. JavaScript
2. C++
3. Java
4. Python

Mobile –

* Possible licensing costs
* Will likely need fewer developers on par with windows.

Tools

1. Apple-HTTP API
2. Buildfire.js
3. Ionic
4. Android Studio

Languages

1. Objective-C
2. Swift
3. C ++
4. Java
5. Javascript

## 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**: Windows would be the best option for development. Windows has all of the software and IDE’s available to it to develop and push software to the other platforms with more ease than the other operating platforms. There are also all-around IDE’s like Visual Studio which come with multiple libraries built in for developing on Mobile and Linux based platforms. For our purpose of developing an application that will utilize API’s it also makes sense to use Windows since we can develop with GitHub and ATOM easily. Allowing us to use Javascript for our API development.
2. **Operating Systems Architectures**: Windows services are very wide ranging a encompass the spectrum needed to develop and communicate in the modern world. The architecture for these systems rely on Graphical User Interfaces for ease of use while also retaining the more complex software needed for deep dives such was Powershell and the new windows terminal app.
3. **Storage Management**: Windows makes storage a fairly simple process with disk partition tools for new storage media and control. Windows also integrates services such as OneDrive for easy cloud storage usage so that your information is always backed up and safe in case of disaster. If you have a Systems Admin, they can utilize the PowerShell scripting ability to quickly set up storage for developers and clients so they never have to even think about it. Our server-side storage can be stored on an SSD array and run in a RAID formation. RAID allows us to continually run our server in the case that one of our SSD’s fail. With a hot-swap option we can replace that faulty SSD while our servers continue running. Our minimum recommendation for this program is to use at least two 1 TB SSDs. One SSD will store our images and the other will store the program software.
4. **Memory Management**: Memory management with Windows will include storing the database or library on a drive and using the “folder” system to store the information needed in the program. Since we will be utilizing a lot of pictures it would make sense to use a logical partition on a drive and using that solely as an area to keep the pictures when the program calls for them. We will recommend a CACHE memory server that brings up the last image pinged so that during a game session the cache server can be pinged continuously as the image is unblurred to the players. This type of memory management allows an image to be quickly called upon when needed instead of constantly accessing the SSD in the storage server which may also contain the programs code in a minimal system use case.
5. **Distributed Systems and Networks**: Our recommendation is that Windows can be used on the servers in the company but will require a systems and network engineer to maintain. There will have to be servers set up which will host the game, host the games database, host the different versions of the game if going that route, as well as hosting other quality of life servers. In order to make sure the game is scalable to traffic you may want to include a load manager, and other services to make scaling easier. One example of a database server would be having the Team and player information hosted in one database while it is called by the server running the game on another. This way the load is distributed between hardware, and you can further scale from there. Windows also has many different IDE’s that will allow the development of client-side and server-side communications to ensure that this program can work on any platform. Developing with HTTP/CSS/Javascript will allow us to accomplish the “GET” “POST” and other API calls needed for our application to run.
6. **Security**: Windows is constantly updating their OS with the latest software to combat the growing security threats in the world. Although Windows is uniquely situated as the most common operating system, it also suffers from that since most viruses and exploits are aimed at the windows OS. As windows begins to open up on opensource development and see the light, they will inevitably begin to see greater security that comes with thousands and thousands of engineers updating a system daily. There are also several great anti-virus’s and other security software out there that you can purchase to protect your equipment with excellent customer-support. Although Windows may have the reputation of being the Operating system that is most vulnerable to attack, if it is properly secured your system will be as protected as any other out there. If we were really concerned with security and the potential of Man in the Middle attacks or snooping we could potentially include a VPN service for our application. The drawback is that there will be a slight loss in internet speed that naturally comes with routing traffic through different nodes before reaching the client. Another point of security would be to encrypt user information on our servers to ensure that should a potential breach happen, criminals could not access our users personal information.