

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 | 09/18/21 | Steven Cruickshank | Laid out design plan for client’s guessing game web-application |
| 1.1 | 9/30/21 | Steven Cruickshank | Laid out further instruction on client-server design pattern under Evaluations. |
| 1.2 | 10/17/21 | Steven Cruickshank | Added final 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 client The Gaming Room requests our help in designing a web-based version of their application *Draw It or Lose It*. The client us relying on us to construct the ideal streamlined environment to roll out this web application. The client requires each game to have one or more teams, with multiple players on each team. Game and Team names must be unique, and only one instance of each game can exist at any given time. Using unique identifiers will accomplish those requirements.

## [Design Constraints](#_2et92p0)

-Scalability: The client is an established company with a high number of users. Software will be designed in a streamlined way to handle their ever-growing userbase.

-Functioning programming across multiple operating systems and web browsers.

-Hardware requirements: We should keep in mind that the client would prefer a low entry point regarding future hardware requirements. The application should be able to run on a wide array of machines to help grow the client’s userbase.

## [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)

See below for class diagrams representative for this web application. We’ll start with the Entity class diagram. You can see the attributes id and name. The entity will represent an instance of a match. Using the id and name attributes, we can compare all new entities to existing entities to ensure each entity is unique.

Let’s take a look at the classes which are inherited by Entity. This is notated with an empty arrowhead attached to a solid line. These classes are Player, Team, and Game. We see the Team Class relates to the Player Class in a “none to many” relationship. Meaning that there will be many players per team. Similarly, the Player class relates to the Team class in a “none to many” relationship. Each of these three classes have an array of operations, and two provide attributes to assist us.

Looking to the left of these three class diagrams, we see the GameService class relating to the prior three classes in a “none to many” relationship. The GameService class will act as the central server for the web application. It has an array of attributes and operations to aid us.

The ProgramDriver will act as our main driver for the program. The black filled arrow shows us that the ProgramDriver is constantly sending synchronous messages to the Singleton Testing class. There is a usage dependency between these two class diagrams. A singleton is chosen as it allows one instance of the game.

**"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** | It can run windows server-side frameworks like .Net in addition to its own. My experiences with XCode have been difficult, but Swift may be more programmer friendly.  Mac OS does offer server-based deployment. Unfortunately, MacOS is notorious for expensive proprietary software/hardware, which will drive up licensing costs to the client. | Linux seems very secure, safe, and very stripped down on the server side. In my experience, Linux is the least user-friendly of all the operating platforms regardless of server/client side. Other than safety, there doesn’t seem to be a huge upside server side.  Linux does indeed offer a server-based deployment method. There are no discernable licensing costs to the client/TheGameRoom, as Linux distributions are free. Linux does require a very specialized knowledge. | There is likely a large talent base for server-side programming in a windows environment. Thanks to the global popularity of Windows OS, there are constantly new and emerging security threats that will need to be properly dealt with.  Windows does indeed offer a server-based deployment method. Windows OS is not free, however it’s licensing costs won’t be nearly as bad compared with Mac OS. | With the affordability of mobile devices, it seems tempting to focus on the mobile market with a web-based app. A major drawback is their size. The user experience drastically changes, especially if you’re trying to access a browser-based web-app on a smart phone.  Android does not offer a server-based deployment method. There are likely no licensing issues given the wide availability of Android OS. |
| **Client Side** | MacOS and its software is notoriously appealing aesthetically. I believe MacOS is safer client side compared with Windows.  Mac OS uses very specific languages (Swift, XCode) which will require more specialization when it comes to developers. Use of native code class wrappers are recommended to retain usage of original Java code. | Linux has an extremely small userbase given the expertise needed to operate it. Given the money needed to hire linux-knowledgeable developers, and the small userbase, it seems like a waste of money to use Linux overall.  Given the relatively small userbase of Linux compared with other operating platforms, the client should consider the extra development costs needed to support this operating platform. Android can run Linux web-based application with not much fuss, however being able to run a Linux web app from iOS without virtualization tools. | As stated, Windows is the most popular OS for all kinds of applications (business, gaming, etc). This is likely the cheapest way to go as far as cost to hire developers, as there are so many with experience in Windows.  Given the large userbase of Windows, there are innumerable tools, frameworks, etc. that will easily aid in cross-platform compatibility, especially compared with Linux.  Should the Client want to expand operating platforms to include Xbox or PS5, this is what I would recommend. | Mobile development is absolutely a different set of skills than development on a traditional OS. It will require specialized developers which means more money, and possibly time.  There will need to be cross-platform programming to make things compatible between Android and MacOS/iOS which will increase development time and expertise. |
| **Development Tools** | I’ve had difficulties with XCode in the past. I’ve heard Swift is friendlier but have no experience with it. I’ve used VIM through the command prompt with success and enjoy its simplicity.  Swift and XCode are the programming languages used with Mac apps. XCode specifically has its own IDE. The dev team will need to be able to handle the difference in languages. As stated, Mac development will increase licensing costs dramatically. | Swift can only be deployed in the Ubuntu distribution. Otherwise, many of the same languages and IDEs can be used (python, C++, Eclipse, Visual Studio, etc)  As discussed, Linux is open to a wide array of IDEs and languages, which makes it ideal for hiring developers in terms of spending. Linux distributions being free helps to cut down on development costs as well. | Windows plays well with a variety of your expected languages and IDEs (python, C, Visual Studio, Eclipse, etc). Windows machines cannot develop in XCode, but I believe it’s possible to code in Swift.  Apart from the XCode incompatibility, Windows plays well with most IDEs and nearly all languages. This makes for more economical hiring. Likely not many licensing costs beyond Enterprise IDE software licensing. | I have experience programming in Android via Java, but I’ve heard of people using Kotlin as well. For an iOS app, I believe one would use Swift.  Java is going to be ideal for an Android mobile application. They were practically made for each other. Licensing costs would be minimal with Android.  Any iOS-based app would need specialized developers with specific expertise in MacOS related programming languages/IDEs, as well as increased licensing fees. |

## 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**: My recommendation is that we use the Windows operating platform. With such a large userbase, it will give the client plenty of room to expand. It also provides a vast pool of developers who will work for cheaper compared to a Linux-specific developer. Would make a relatively simple transition to a mobile app as well.

*Updated 10/17/21:*

I’m recommending this application be run off a windows server. This will allow access via web browser (both PC and mobile). In doing so, this program will provide the most bang for its buck. You will immediately get the largest userbase of PC/mobiles users all by using development techniques that are not particularly specialized. This means the client can keep development costs much lower than say, an OSx server.

In addition to the reasons above, this will also offer the most promise for potentially expanding across more operation platforms. For instance, Playstation and Xbox operating systems will seamlessly link to a windows server via the appropriate APIs.

1. **Operating Systems Architectures**: This operating platform will be a Windows 64 bit architecture. Many legacy computers were using 32 bit architectures, however all newer Windows machines are using 64 bit architecture. The main difference being the OS can access 2^64 of memory, whereas the old machines could only access 2^32 bits of memory.

*Updated 10/17/21:*

This web application will utilize a 64 bit Windows OS architecture. This architecture

has a design that is layered, and includes several components. The two most important components are the Kernel mode and the User mode. Simply put, User mode will pass I/O packets between layers. Kernel mode will make sure the user isn’t accessing or modifying critical system infrastructure.

1. **Storage Management**: I recommend using a database to store the necessary images using the JDBC API or similar database tool. Any user information (name) will be wiped after the end of each game.

*Updated 10/17/21:*

An important distinction between memory management and storage management is that storage management will refer specifically to “longer term” memory, whereas memory management will refer to short term storage of bytes.

In looking at the client’s request to pull from a single repository of image files, my recommendation is that we use a single-level file directory in which to hold this repository. While a single-level file directory does have drawbacks (namely importance of using unique names), it seems to be an ideal method for this project.

The client will mount the virtual file system which is provided by the server app. This will provide the client access to these image files so that they may play the game. At its most basic level, the list needs to be capable of the following: searching for a file, list the files, and transverse the list of files.

1. **Memory Management**: The key to memory management in this program will be cleaning up what is created dynamically. Windows will use a combination of symbolic, relative, and physical addresses when referring to internal memory. We will emphasize dynamic memory creation in order to save on static memory.

*Updated 10/17/21:*

An important distinction between memory management and storage management is that storage management will refer specifically to “longer term” memory, whereas memory management will refer to short term storage of bytes.

As stated, the data pool is stored as a static group of image files. Using a sequential access system will suit this project fine. As the name implies, data is read into memory one after the other. While incredibly common, it is one of the more resource intensive systems of access, its effects on this application will be minimal given its size.

The file system interface will help with determining how much memory will be needed for the data. Upon implementation, I can use open(), read(), write(), and close() to access these file’s data. Step two is to implement the Virtual File System interface, which is an API available in Java. As the name implies, it will help set up a single directory virtual file system. This is virtual storage on the server. Some other perks of VFS include that it will allow several instances of the interface to exist at once. It also uses unique identifiers for the data.

As one can imagine, these image files are not going to be the same size. As this data is being accessed into memory, the tail ends of some blocks of data are going unused. For one or two files, this isn’t too much of a concern, but as the amount of data grows, more and more memory will be left unused. I have a few thoughts as to better manage this memory and reduce the amount of fragmentation.

Packing bits using bit shifting could be a useful way to deal with possible internal fragmentation. More importantly is to implement first fit, and best fit algorithms. Best fit will place the data in the smallest block of data that’s available, while first fit will find the first available space (starting from the beginning). Implementing these algorithms would go a long way in preventing huge chunks of unused data.

1. **Distributed Systems and Networks**: My recommendation is that we use a TCP/IP connection to communicate between users playing the game. The web app will be distributed among Windows users who will be connected through the internet. Unfortunately this game will be reliant on the internet to connect users. If a mobile app is ever developed it will be reliant on Wi-Fi or a Cell Phone Network in order to connect users.

*Updated 10/17/21:*

The web application will be hosted in the client/server model for best performance. This allows many instances of the server to be simultaneously viewed by multiple client instanced. It is by far the simplest and most effective method of distributing this web application.

There’s also benefits in using a Client/Server distributed system model. Should there be any loss in connectivity, the Client/Server handshake is able to seamlessly pick up where it left off via data verification between mounted virtual file systems. Losses in connectivity can occur when the server goes down, or when a mobile user enters a dead zone for example.

1. **Security**: Given it’s a Windows machine, there will always be new security risks that need to be addressed. In addition to vigilance, we can use a combination of common web app security mitigations, like preventing SQL injections, data authentication, and avoiding using components with known vulnerabilities.

*Updated 10/17/21:*

An integral and very basic part to proper security on this web app is to properly set up domains and read privileges for each user. While it might not seem like much, proper allocation of privileges is an easy pitfall to avoid. As one can imagine, granting write privileges to the client (or even worse, admin rights) is a very easy way to have a security breach. The system-wide file table will contain information regarding privileges, etc.

Using the Virtual File System java API will also aid in keeping data secure. It safely provides open(), read(), write(), and close() privileges as necessary. Some other helpful perks of using the VFS API is that it allows several implementations of the interface to coexist among clients. It also allows for unique file ID’s to be used (via VNode). Writing sound and simple code is an incredibly easy way to ensure there are no security flaws. In using the appropriate API, we can be sure the program will be coded in the simplest, but effective way possible.

As stated above: Ensuring that developers avoid components and APIs with known vulnerabilities is another easy way to avoid security exploits.