

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 | <07/21/2024> | Dean Youngblood | Initial Document creation |
| 1.1 | <08/04/2024> | Dean Youngblood | Evaluation review |
| 1.2 | <08/18/2024> | Dean Youngblood | 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 Gaming Room needs a software solution that will gradually render images from a library of stock images over a 30 second period. This software should allow one team to guess during this 30 second window before moving on to the remaining teams and giving them a 15 second window to guess the phrase, title, or thing that the image is depicting.

## Requirements

This application will have the ability to have one or more teams involved. Each team will have multiple players assigned to it. The game and team names must be unique such that team names can only be used once and users will be prompted to select a different name if their chosen name is already in use. Only one game instance can exist in memory at any given time, such that each instance of the game has its own unique identifier.

## [Design Constraints](#_2et92p0)

Building Draw It or Lose it as an online application requires the utilization of Real-Time Communication, utilizing sockets for a constant two-way flow of information. The concept of RPCs (Remote Procedure Calls) is also relevant here so that everyone playing the game receives a consistent view of the game state such as the active rendering to be guessed, the score, and the timer. The rendering of the image to be guessed should be handled on the client side while the timer, score, and related actions for answering the questions should be handled on the server side of this web application. It’s also important to keep in mind that multiple instances can occur simultaneously and need to be handled separately. This is why the singleton pattern, and the iterator are used for this project.

## [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 UML diagram provided, we can see that the GameService class has a zero-to-many relationship with the Game class which has the same relationship with the Team class. This is again followed as a 0-to-many relationship between the Team class and the Player class. This means that there can be any number of instances of each class in relation to each other. The object-oriented principle of inheritance is also clearly depicted as Game, Team, and Player all inherit from the Entity class. Encapsulation is demonstrated via “-“, showing that there are private fields in several of the classes. Polymorphism is also being utilized via overriding the toString method from Entity in the Game, Team, and Player classes.

**"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** | Pro(s): MacOS has very consistent OS and hardware integration, ensuring compatibility. It’s also typically seen as easier and more intuitive to set up.  Con(s): MacOS servers are quite costly. | Pro(s): Secure, cost effective, and efficient. Linux can run on a broad amount of hardware and typically uses less resources than MacOS or Windows.  Con(s): Setting up Linux servers can be more difficult for those not familiar with CLI. There is also limited commercial software support and the broad scope of hardware being used can cause some compatibility issues. | Pro(s): The best remote desktop support of the 3 main desktop operating platforms. It’s also widely available and is familiar to most people.  Con(s): Windows updates can often cause issues or breaking of both software and hardware components. Windows typically has worse performance as well due to a larger overhead. | Pro(s): Assuming we are talking about cloud hosting here as I have never heard of someone hosting directly from a mobile device. There is typically great customer support and recovery options with cloud hosting, and there are plans to scale to your needs.  Con(s): Cloud hosting can sometimes be slower depending on traffic and bandwidth allocations. It can also get quite expensive depending on processing demands. |
| **Client Side** | MacOS users are most likely using the Safari browser which utilizes WebKit. This can cause some differences in performance and would require Apple hardware to be involved on the development side. However, there are only a limited number of OS versions currently in use to be considered. | There are hundreds of different Linux distributions available to clients. However, most of them come packaged with either Firefox or some form of Chromium. Some uses might experience poor performance depending on their browser settings such as hardware acceleration, etc. | Much like Linux, costs could increase due to the need for testing and development in many varying environments due to different hardware and operating system configurations. There is also the added workload of needing to correct bugs across constant Windows updates. | Developing for an Android client will likely involve Kotlin which is very similar to Java. However, for iOS you would want to use the Swift programming language for optimal performance. This means you likely need two different development teams, one for each port. It’s also important to ensure that the UI is tailored to small screens and touch interactions. |
| **Development Tools** | HTML will be used across all operating systems when developing a web app. All of them will also likely use python or JavaScript for the back end of the application. MacOS will likely require a Swift skillset and potentially use the legacy Objective C language. This means X code will need to be used as an IDE, as well as an Apple developer license to certify and publish the application. | Both Linux and Windows will likely take advantage of electron for simplicity and development cost/speed. This means the application will primarily use JavaScript. | Like Linux, electron is likely to be used here. There is also the option of going a more performance-oriented approach and using C++ or .NET. | iOS will likely use Swift in the X Code IDE. Android apps will likely use Kotlin in the IntelliJ IDE suite. There are also tools available to port aspects of the software over from Android to 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**: I would recommend using Android as the primary development target platform. There are many tools available to port from Android/Kotlin to other platforms with low performance hits or UI issues. This would benefit most from being cloud hosted as needs will vary greatly depending on how popular the application becomes. It’s important to ensure that there are enough resources to run smoothly even with (hopefully) millions of people accessing it.
2. **Operating Systems Architectures**: Assuming that AWS is utilized, there will be a serverless architecture in place being managed by AWS themselves. Other platforms should also have similar features
3. **Storage Management**: Cloud platforms utilize scalable storage that shrinks or expands as needed, sometimes automatically. AWS in particular, also provides fully managed SQL databases for our storage needs.
4. **Memory Management**: We can opt to have memory management fully automated to dynamically allocate the necessary memory to keep our application running smoothly.
5. **Distributed Systems and Networks**: By using cloud hosting we can take advantage of tools offered by hosting companies to create and manage our RESTful APIs. Additionally, a cloud hosting services will utilize multiple nodes as our server. This means we can add or remove nodes as needed to scale the application. Our application can also be distributed across the nodes to ensure smooth operation without maxing out resources on a single process or machine.
6. **Security**: While our chosen cloud hosting platform should handle most security concerns for us, it’s important to make sure that industry standards and best practices are following. All data should be encrypted, and HTTPS should be utilized for access. The SQL database should utilize input validation, and all keys need to be secured to ensure that no one is able to hijack our cloud hosted app or access sensitive information. Any sort of log-in to the cloud platform should also use some sort of MFA such as an authenticator in addition to the encryption keys.