

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 09/22/2024 | Tristan Maloy | Updated Executive Summary, Design Constraints, and UML Domain Model |
| 1.1 | 10/06/2024 | Tristan Maloy | Updated Development Requirements |
| 1.2 | 10/08/2024 | Tristan Maloy | Updated Executive Summary, Design Constraints, and UML Domain Model |
| 1.3 | 10/20/2024 | Tristan Maloy | Updated Recommendations and Development Requirements |

**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)

Creative Technology Solutions (CTS) is working with a new client, The Gaming Room, to develop a web-based game that serves multiple platforms that is based on their current game, Draw It or Lose It, which is currently only available on Android devices. The game will need to support multiple players and platforms. The game will also consist of four rounds that last one minute each where the player/team must guess what an image is as it is slowly rendered to completion. If that player/team does not guess correctly, the next player/team has a 15 second opportunity to guess themselves.

## Requirements

1. The game requires one or more teams to play.
2. Each team will contain multiple players.
3. Game and Team names must be unique
4. Only one instance of the game can exist in memory at any given time. This will include unique identifiers for:
   1. Each instance of the game
   2. Each team
   3. Each player.

## [Design Constraints](#_2et92p0)

The Gaming Room would like the game to run on multiple platforms so a programming language that is universal to the chosen platforms will need to be utilized. Having one programming language will decrease programming time, which will allow the project to be completed in a smaller time frame. Singleton design patterns will need to be utilized to ensure there is only one instance of the game at any given time. Iterator patterns will need to be utilized to ensure that game names, team names, and player names are unique and not duplicated.

Since the program has already been created on an android OS, adapting that code to be usable on other OS will be a big design constraint. If the app is going to be cross platform, there is the added constraint that each OS will have to work in unison and share not only code but instances of the game. Since only one instance of the game can exist, it needs to be compatible with all OS at the same time.

Security is a design restraint that has a heavy impact on the safety and privacy of players’ user data. Since the game will host multiple players and teams, each player needs security to prevent access to their data and hardware.

Users will need to create accounts to allow authorization and authentication for the application. This will also help with security. Accounts should be compatible with all OS to promote cross platform, so should the requirement be a “The Gaming Room” account to allow access to the game servers?

## [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 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.**

The UML Diagram for the app consists of 7 classes: ProgramDriver, SingletonTester, Gameservice, Entity, Game, Team, and Player. ProgramDriver is the main driver class that is used to create games, teams, and players. ProgramDriver utilizes the GameService class to create the game, teams, and players.

The Game, Team, and Player classes all inherit the parent class Entity which gives each class access to Entities 2 private attributes, id and name, as well as accessors for the variables. which demonstrates polymorphism. This also ensures that each class has a unique id and name since it inherits the variables from Entity.These classes, as well as Gameservice also have private attributes and public methods and mutators which demonstrate encapsulation and abstraction. An iterator pattern is utilized for Game, Team, and Player as well to ensure every name and id that is attributed to the class lists are unique.

GameService uses a singleton pattern to ensure that only one GamerService class may exist at any time in memory. GameService also uses a private constructor to prevent more instances from being created. There are also accessors for the Game, Team, and Player classes.

## [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** | Mac has notoriously high security features that could help protect servers for hosting web-based applications. Generally, MacOS has stayed consistent in terms of software so compatibility amongst different versions of MacOS would be easy to accommodate. Virtualization can be done on MacOS; however, it would require a higher amount of ram than a base Mac has so it isn’t recommended. Licensing fee for Apple’s Developer program is $99/year.  Prices for a dedicated server vary from $25/month to $259.99/moth and potentially more. These numbers are based on hosts such as [www.macminivault.com](http://www.macminivault.com) and [www.macincloud.com.](http://www.macincloud.com.) | Linux is open source, so most resources, tools, and software programs are free and provided through the community of users. Security is customizable and easy to adjust. Linux has built-in user privilege management which will aid in the authentication and authorization of users on the server. Virtualization is key with Linux, as there are compatibility issues with file types. Linux can run a VM for Windows and MacOS which will allow better compatibility server-side. Linux does not have a licensing fee. Linux supports PHP, Perl, Ruby, and Python for web hosting. A popular cloud-based storage server is pCloud which has free 10GB storage as well as encryption for client-side data. | Windows is cheaper than Mac, however, it is prone to more cyber-attacks as the security is not as good. Windows has more compatibility with commercial software and resources.  For a dedicated standard server via Windows Server 2022, the price for a license is $1069 while for a datacenter license the price is $6155. Both prices are for 16 core licenses. Out of the four OS, Windows is the only OS that is not UNIX-based. | Mobile devices expected to host a web-based software application would need to be a lot stronger than current mobile devices to run properly and sustain thousands of players. Mobile devices are also more prone to cyber-attacks than the other three options. As a result, mobile devices should not be used as a server. |
| **Client Side** | Browser compatibility will be a big consideration as MacOS has its own browser with safari. Internet Explorer can be accessed through a virtual desktop but not directly on the MacOS. Browsers such as Google Chrome and Microsoft Edge are supported on MacOS. MacOS is the second most popular desktop OS. | Since virtualization is popular amongst Linux users, browser compatibility is easy to consider when using Linux. Natively, Linux is compatible with Firefox, Chrome, Microsoft Edge, and multiple other lesser-known browsers. Linux is not generally used for gaming however, since there are some compatibility issues with files between different OS. Out of the three desktop OS, Linux is the least popular. | Windows has the most compatibility and can virtually simulate other OS to ensure browser compatibility. Natively, Windows can run all browsers, with Safari being the only tricky browser since Apple no longer supports the development of Safari for Windows. Windows is the most popular desktop OS when compared to MacOS and Linux. | An in-web web-based application directly though the web will need to be able to support the different browsers, screen sizes, and hardware limitations. Mobile devices can utilize a multitude of web browsers, with the only exception being Safari because there is no official safari app for android. However, most of the population always has access to a mobile device so developing a program for a mobile device would be beneficial to get an application in more users’ hands. |
| **Development Tools** | MacOS runs on the Apple programming language Swift. The IDE that is most used is Xcode which is free to use with a normal apple account. There is a paid Apple developer program for $99 that also unlocks full access to development tools, testing, and services. A C-based language could also be utilized for better compatibility with other OS. | Visual Studio, Eclipse, and IntelliJ are compatible with Linux. Since Linux is open source, there are many different development tools that can be used for development. These development tools won’t be the same as the ones used on MacOS or Windows but will be compatible with the same languages and perform similar actions. Bluefish is an IDE that supports a multitude of languages. The best programming languages for Linux include Python, C++, Java, and Swift, just to name a few. To stay compatible with multiple platforms without the need to code in multiple languages, A C-based language could be used to cover all OS. | Windows can use many different IDEs from Eclipse and Visual Studio, to PyCharm and IntelliJ. Windows can be versatile with programming language as it supports most languages, and if not, a virtual environment could be utilized to use other languages. | **Mobile devices are not meant to be used as development platforms.** Developing for mobile devices using one of the other OS would utilize programming languages such as Swift, Kotlin, JavaScript, and JAVA. Swift is the official language for iOS development, Kotlin for Android, and JavaScript and Java are used for web and cross-platform mobile development.  iOS development can be done in Xcode using either the free application or the $99 subscription while Android utilizes Android Studio which is free. |

## 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**:

To keep development requirement costs low, it is recommended to develop Draw It or Lose It using Linux OS. Linux has no licensing fees which would allow more money to be utilized in other areas such as dedicated servers, security, and storage. Linux is open source and has an active community for support which promotes innovation and a continuous stream of new features and tools that can be utilized for development. It is also compatible with a large variety of programming languages which makes it versatile for developers with different backgrounds to create on.

1. **Operating Systems Architectures**:

The bulk of the program will be contained using a web-based server that the user will connect to via the application using RESTful API on their preferred system of choice (windows, mac, mobile etc…). The server will contain access to cloud storage to access all the images while natively having access to all the processes required for the game while the application (client side) will connect to the server and contain enough RAM to sustain a lag-free environment while loading pre-generated images from the memory. If the server-side lag is an issue, since there will be extra allowance in the budget by using Linux, multiple servers (i.e. dedicated regional servers) can be created to sustain a larger number of users.

1. **Storage Management**:

The server will contain permanent storage for all the images required for the game. Draw It or Lose It will have 200 8MB images which will require 1.6GB of storage, plus enough storage to contain the processes for running the game. The server can utilize cloud storage, such as pCloud which has 10GB of free storage and is compatible with android/iOS and can be accessed from all web browsers.

1. **Memory Management**:

It is recommended that the application requires enough memory to load enough images to complete a whole game without having to contact the storage to reduce waiting time and lag. At the beginning of a round, the application will load enough images from the cloud storage into the memory, and throughout, the game utilizes paging to access the images in the memory which will get rendered continuously to the user. If the system starts to run out of RAM, the application can use swap space to manage storage and memory. This will take the lesser used processes to and from the disk when needed to free up space in the memory.

1. **Distributed Systems and Networks**:

RESTful APIs are necessary for communication between the different platforms the application will be playable on, and the server that will be developed on Linux. The client can put a HTTP method request to the server which the server will interpret and respond with the process or necessary files.

A microserver pattern can also be utilized. This will help to develop a mobile interface and a web interface for the respective devices the application will run on and then connect to the same microservices that the two interfaces will share. This ensures that the program is scalable to help prevent reductions in performance.

Utilizing dedicated regional servers to allow an overflow of users in case the server is congested will also help to reduce lag for the game.

1. **Security**:

Linux has built-in user privilege management which will aid in the authentication and authorization of users on the server which will provide security on the server-side of the development. On the client-side, profile creation with password protection and passkeys will be utilized to only allow users that are authorized to play the game access. The utilization of pCloud can also add client-side encryption for added security.

Throughout the entire program (server-side and client-side) the principle of least privilege will be enforced to ensure that users and systems have enough access to perform their required tasks. This would allow users to only have access to processes and tasks that only users should have access to such as: creating an account, playing the game, and selecting settings such as sound options. This, in addition to utilizing RESTful APIs with @AUTH, @RolesPermitted, and @PermitALL annotations will increase security by only allowing users with admin roles to alter the game.