

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

[Table of Contents](#bookmark1)2

[Document Revision History](#bookmark2)2

[Executive Summary](#bookmark3)3

[Design Constraints](#bookmark4)3

[System Architecture View](#bookmark5)3

[Domain Model](#bookmark6)3

[Evaluation](#bookmark7)3

[Recommendations](#bookmark8)5

## [D](#bookmark9)[ocument Revision History](#bookmark9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/18/2022 | Luna Deighan | Created Document and filled out Executive Summary, Design Constraints, and Domain Model. |
| 1.1 | 10/09/2022 | Luna Deighan | Completed Evaluation Section of Project Design Document. |
| 1.2 | 10/16/2022 | Luna Deighan | Completed recommendations section of project design document. |

**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.

## [E](#bookmark10)[xecutive Summary](#bookmark10)

The Gaming Room is seeking to employ the services of Creative Technology Solutions to develop a web-based game that services multiple platforms based on Draw It or Lose it, their currently Android exclusive game. This application will ensure that Draw It or Lose It reaches a wider audience via a streamlined service with accessibility features for many different customers.

## [Design Constraints](#bookmark11)

The game must support multiple teams being involved within, but also have the ability to have a solo option for a singular team. Each of these teams will have multiple players assigned to them and each team must have a unique name identifier. There must also be support for multiple games to be stored and, as mentioned before, must have the ability to have any number of teams involved. Each of these games must also have a unique name identifier. Finally, only one instance of the game can exist at once and must be called to play within.

## [System Architecture View](#bookmark12)

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.

## [D](#bookmark13)[omain Model](#bookmark13)

In the UML class below, the entity class can be seen as the parent to Game, Team, and Player, each of which inherit general properties of it. The Player class is simply holds a name and id and is the unique object for each user logging into the system. The Team class has the main feature of a list of Player Objects, which serve as the team members for each specific instance of a Team. The Team can also add players to it if needed. Next, the Game class has the main feature of a list of Teams that are playing within it. This class also offers the ability to add Teams to an instance. All of these Games funnel into the GameService Class, which contains a list of games and is a singleton class where only one instance can be created at a time. This call can give information on any game that it contains, a specific stored Team or Player, and can add in additional games. Finally, the ProgramDriver and SingletonTester files run the program itself, with the ProgramDriver file getting the current instance of GameService and running the code and then using the SingletonTester file to ensure the GameService instance object is properly functioning.

**"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.**

## [E](#bookmark14)[valuation](#bookmark14)

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 does not have the ability to run server side applications any more due to its discontinuation of the MacOS server architecture in April of 2022. | Linux as a server is the most popular choice due to its Unix like properties. It is free and open source, meaning the cost will be down greatly, has end-to-end encryption making data transfer immensely safe, and has great integration with almost every programming language out there, meaning the server could be programmed in any language. However, there is some lack of long-term support and interface with professional programs that can cause problems alongside it being a little more complex than certain other choices and having a relatively obtuse update process. | Windows is typically used for client-side operations, however in hosting a web application it has a few unique features. It is much more beginner friendly than its unix competitors and supports a large amount of third party applications alongside having a great support system should problems arise. However, it is immensely expensive with a per-user pricing model and has been prone to quite a few security errors. Due to it not being open source, the way the system works isn’t fully known by the public beyond documentation put out by Microsoft. | Mobile Devices are unable to host a server to any reasonable capacity beyond an android file storage system. There simply is not enough computing power contained within one to do so along with a full lack of software pertaining to server-side hosting. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Client Side** | Some things to take into account with building client side applications for Mac is the ecosystem being much more closed off than other computer/desktop OS’s. However, Mac’s are also able to access a desktop browser and traffic can be routed through this, meaning to save cost and time, the application could be fully web based. For full integration however, it may cost a little more and require the expertise of someone who specializes in Apple/MacOS development specifically. To support multiple types of clients, it would be pretty simple on this platform however, as all clients with Macs will have the same operating system and therefore won’t need any specific tweaking. | Linux would be one of the most complicated operating systems to develop a client side application for due to the mass variety of the user base. Because there are hundreds of linux distributions, a very generalized application must be made if it is not fully web/browser based. However, development is relatively simple due to the platform interfacing extremely well with almost every language. But, even with it being easy, Linux has the lowest user population of the 4 listed Operating systems and therefore will not bring many new clients. With those few clients however as previously mentioned, supporting multiple types becomes complicated as they may be using completely different distributions of Linux, leading to the application built looking great on some users’ computers but awful on others. | Windows is the most used Desktop OS and as is likely the easiest to develop a client-side application for due to the breadth of knowledge from the many apps already on the market. First, it can run executable files, making distribution of a client based app simple and easy as most languages that would be used for a desktop app can very simply be converted into an executable at the end of development. All modern browsers work perfectly with this OS as well due to its popularity and are frequently updated, so accessing the application will always look good even if it is just through a browser. Finally, for multiple types of clients, it is immensely easy to upscale due to the ease of use on this OS and the homogeny of the user base. | Mobile is the most frequently used way to access the internet and will be crucial for the applications success. However, this platform also has the largest variety in user population as there are many different OS systems and even within those OS systems, there are many different devices running them. Therefore, many different versions of the web app must be designed to be viewed on every screen type/size in order to look good and be accessible by all users. This will require the expertise of mobile designers as designing a responsive and stylish UI for a mobile device is an entirely different skillset than doing the same for a Desktop application. The cost would also be higher due to having to make multiple mockups and versions of the application to be viewed for each device size (phone vs tablet) and building them out. However, once these items are built, handling multiple users should be simple once again due to the application being web based. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Tools** | Some programming languages and tools used to deploy on this platform specifically would be things like react native, swift, and the electron package for node. React native and/or electron could create a direct application that pulls the web app into the desktop version of a Mac and uses many of the specific qualities MacOS has to boost the user experience. Some other tools for deployment would be Xcode of testing and deployment and any sort of disk imaging software to create .dmg files for distribution. Other than that things like HTML, CSS, and JS can be used as the bread and butter for the main web app across the board (I will not be bringing this same part up in every Desktop section for brevity but know this note applies for Windows, Linux, and Mobile too). | The best languages to use for building a Linux app are most likely C, C++ based or something like Perl /Python. C++ may be the best choice to save money and create a localized desktop app to read the site. However, besides languages not much else is necessarily needed for deployment as Linux can run most any deployment types due to its open-source/flexible nature. It also does not have a specific IDE required due to the languages not really being OS specific. | Windows would best run things like C or C++ as a desktop app, but could also use things like Electron and React Native for a much faster build that integrates the web directly, therefore requiring less programming overall while also providing a more streamlined access process for the user. IDE wise there also isn’t really anything specific needed for this OS as there is not real language specificity required. | Mobile devices are the most complicated Development Tools wise due to needing to create multiple distributions of the same application on the client-side app, whether or not is is integrated directly to the OS. With Web design specifically, something like Adobe XD or Figma would be required to build out the different screen sizes (this would also be required in the other OS’s, but much more so here). If a native app were required to view the webApp and access more of the phones features, something like React Native or Swift would have to be used as the main language (Swift being iOS specific). When developing for iOS specifically, if an application to interface is developed, then Xcode as an IDE is a necessity due to deployment and testing. For Android, their development suite, Android Studio is also a must, however it seems as though the company has an android app already built and deployed. |

## 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 believe the best choice of operating system to host The “Draw It or Lose It” servers on is going to be Linux. I believe that the power behind the server distributions of this OS is extremely powerful and has a great capability to be scaled up to any amount of users for a relatively inexpensive price in comparison with its competitors. The Gaming Room will also save OS licensing money as Linux is a completely free distribution, the only real cost to this OS will be from its development, maintenance, and physical or cloud architecture costs.
2. **Operating Systems Architectures**: The system architecture that should be used for this program should be the model-view-controller pattern. This can very easily be deployed on the Linux Operating System through a variety of languages and will allow for a large amount of control from the backend and low effort from any client receiving the program, therefore expanding the reach of the program to as many customers as possible. This architecture will allow the user to interact with specific webpages generated by the program that directly interface with backend objects, allowing instances for each game to be created and run simultaneously while separating the data based on instance.
3. **Storage Management**: I believe the best storage management system to be a direct access file system in which files are able to be read and written directly by the server and any administrative users. There are two main reasons for this, the first being efficiency of data access by the program allowing for many instances to exist of the game at once accessing the files quickly and effectively, and the second being the ability to change the data stored within the file system directly as time goes on. This second feature is important as with a game that relies on repeat customers playing with their friends, updates to the content a user is able to interact with will eventually be necessary to ensure longevity of the project. Allowing direct access to the files, mainly through a CLI in Linux for administrative users such as developers, could be a quick fix to this down the road and ensure the game retains its players for years to come.
4. **Memory Management**: Indexed allocation would be the best form of memory management for Draw It or Lose it as it works directly with direct access storage management and allows the user to quickly and effectively access data that is larger than a few blocks, which most long term stored data is. There is some inefficiency with the smaller variables such as player/team score and stored guesses/prompts when it comes to memory utilization, but this allocation method allows for the most effective use of our storage especially considering the access rates needed if this project becomes highly scaled, which is ideal.
5. **Distributed Systems and Networks**: Using a RESTful API as the distribution system for Draw It or Lose It would prove highly effective for a variety for reasons. First and foremost, it would allow for access to the application on any device that has access to an internet browser, and in todays day an age that means anyone. This reaches an immensely large customer-base while also staying cost effective as it means development doesn’t have to be directly tweaked to each and every user’s device, but instead broadly created to the dimensions of devices themselves. Next, it is a decoupled system in which the client and user interacts separately from one another, therefore allowing multiple clients to request from the server and interact with it without overloading the server or creating a high overhead cost for server processing power. Finally, it allows an extra level of security as users are not directly interfacing with the server itself, but instead individual pages or views (as mentioned in the architecture section) and therefore allows the program to have less of a chance of hacks or leaks.
6. **Security**: There is in general in Draw It or Lose It, a general lack of personal user information required to interact with the application, but this does not mean that security can be thrown to the wayside. The first security protocol to be implemented is an SSL certification for the application ensuring that all traffic between the client and server through our aforementioned MVC RESTful API is secure. This means that any data sent is secure from hacks or injections that may infect either the users device or the servers on which the API is held. Another form of authentication that can be used will be encrypted passwords for instances of a specific game if they are to be private. These will allow a user to input a password and then only if others are able to input the same password will they be able to join said match. These will be encrypted to ensure they are not stolen and discarded after the instance of the game ends to ensure the password cannot be taken afterwards if it is used elsewhere. Finally, on the server-side of the application, a firewall will be incorporated with tunnels for API interaction and VPN password protected user access for developers and administrators, therefore allowing the clients of The Gaming Room to freely access the specified endpoints created within the application, but prevent any unwanted access to other ports that may lead to a security breach.