

# 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 | 07/07/23 | Phillip Cabaniss | Added functionality for the Player, Team, and Game classes to add players or teams. Added a class to implemented throughout program to hold data, as keep to the singleton design pattern. Added a few functions to GameService as well. |
| 1.1 | 07/17/23 | Phillip Cabaniss | Fixed some errors in my report. |
| 2.0 | 07/21/23 | Phillip Cabaniss | Final revision, adding more details on specific OS’s and their distinctions. |

**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 wants help to develop a web-based game “Draw it or Lose It” only currently available on Andriod. The premise of the game is that application will pull stock images from a database and display them as clues. Then, the players will try and correctly guess the image. There will be four rounds, one minute each in each game. If one team runs out of time, the other has a chance to guess it in 15 seconds. The Gaming room needs functionality added to allow the implementation of multiple teams, each containing multiple players. Games, teams, and players must all be unique.

## Requirements

* Create functionality to add multiple teams
* Create functionality to add multiple players to a team.
* Check for any duplicate teams
* Check for duplicate players on a team
* Cross platform, web based functionality.
* Make sure a singleton design pattern is implemented.
* Working with employees to help them understand the environment

## [Design Constraints](#_2et92p0)

Usually implementing functionality like this can be difficult, but since it is in early development it shouldn’t be much of an issue. The game is currently only on Android, this means we must construct a web application from scratch that runs on both OS’s. We will need to implement some new functionality to the game, team, and player classes. Since there will be repeated use of variables, we will need to create a new class to be implemented into the others that hold data. Lastly, we will need additional web server space for use across multiple platforms. This includes both storage and hosting. A variety of online businesses like Google Firebase is an option that offers all these services. However, the more users, the more expensive it will be for hosting. It is important to consider the volume of users against server and hosting costs in the beginning.

## [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)

Here we can see we made some changes to GameService by adding a few functions and variables. These will allow us to keep track of the team and player ID’s and hold the data in a single place instead of all over. This is a great example of inheritance. All this data will be stored in a single class Entity that will be implemented into the Game, Team, and player classes. This is also called a 0-to-many relationship. Where one class can be implemented to many classes as a sort of starter class to hold basic information. This allows us easily search and check existing teams and players for duplicates. This will also help lay the foundation for future features you would like to have, while maintaining encapsulation and portability.

**"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)

In my opinion, this job is very doable. The biggest issue is the coaching of staff on Apple implementation and app publishing. If they are familiar with the Android process, it shouldn’t be too difficult for them to pick up. There aren’t many design constraints that would halt production, so my evaluation of the project is that it can be done with little requirement. The ones needed will be listed below.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Using Mac, you would be using the MAC OS server, which comes with its own built-in Apache Webserver. However, it was discontinued and integrated into the OS. Now you can buy hardware for server-side development like the Mac PRO, but that starts. At around $6000. It also offers server-specific features like file-sharing, calendar, contact/email hosting, and offers a variety of third-party software to help server functionality.  **Cons:**  While offering 3rd party options, it can be difficult to integrate other OS’s applications unless using iOS products making it pricier for hardware. | Linux comes with built in security and is free to use. Being open source, many developers continually contribute to the development of the OS. Linux a variety of different servers like Ubuntu, Debian, Fedora, and more. The costs usually range from $250-$1300+ per year. Features include high-level language compilers, access to GUI, and has a variety of distribution options.  **Cons:**  It takes a lot of knowledge to use Linux systems, so while it may be free, it may cost you in time. Being open source can also come with its fair share of bugs. | Windows is widely used and offers a variety of tools like data-storage, virtual memory, and great third-party application support. There are also many versions to choose from. You can use Amazon Web Services for hosting, or the built in Microsoft Azure. However, this can add expenses. Anywhere from $500 - $6000+  **Cons:**  Since it is so widely used and familiarized, it is more susceptible to attacks when compared to other OS’s. Windows also requires user-based licensing for distribution. | Mobile devices are a great device for running webapps. It is cheaper to build. Mobile web apps also offer more extensive tools for customizing the UI. Uses universal languages like JavaScript to make it cross-compatible on different mobile OS’s. Typically, the server is handled by a 3rd party like Google Firebase or AWS. If it has cross-platform functionality. You can expect to pay $70 - $320 per month depending on traffic.  ­­­  **Cons:**  To use a web application like Draw it or lose it, internet is required. This also means that depending on network traffic, the application may run slower than usual. |
| **Client Side** | Mac is going to be best suited for Mac users. Apple comes with a wide array of support and can easily distribute work across multiple machines.  Like mentioned earlier, Mac is also more secure and offers a variety of options for customizing the UI down to the letters in a button. While the Mac OS is different from your typical Windows system, it shouldn’t require extensive training to use. | Mentioned before, Linux is open source and offers multi-user, multi-process, and multi-thread operations. This makes it a great choice for team development IF the team is familiar with the Linux system. It can also be more difficult to customize the UI. | Microsoft also provides great support for development and users. It can scale easily, and development is usually cheaper after licensing costs. Windows also offers great UI customization and options for development. Since it offers better 3rd party support, it makes testing smoother across different web browsers. | Mobile client-side development is different. Mobile platforms are not built for hosting server side of applications. Usually, the back end is coded, and the front end is worked out by whatever OS is running it. While this may expediate production, this can lead to platform-specific bugs that don’t show up until testing on said platform. It would also depend on the user to update their app on their own device as opposed to update the server side. Development can also offer great customization features for each specific mobile OS. |
| **Development Tools** | The best Mac development is going to happen in XCode. It is tailored for any iOS device and offers live rendering of applications on emulators. You would need to know the Swift coding language, very similar to Java, or just stick with Java. I would recommend using Homebrew for a package manager and installer. Cocoa Pods is used for the backend and must be installed in iOS applications. There are enterprise licensing fees with development, but it is a yearly cost around $300 and is not required while developing, only during the publishing phase. On Apple products, the main browser is Safari, but other browsers run just fine. Testing on them shouldn’t be an issue. | For Linux, IntelliJ IDEA and Eclipse are going to be your best bet. Mainly for use in Java, and C++, but other languages that can be used are python and JavaScript. Using Git for version control and widespread deployment. Docker for image hosting, and Visual Studio Code will make it easier for cross-platform production. While Linux is free to use, you will still need to pay for licensing. Typically, Red Hat Enterprise Linux (RHEL) is used and can cost around $2500 - $12000 per year depending on use. | Windows is more lenient on development due to such a wide use. Typically, you will use an IDE based on language but a great all-around option for windows is Visual Studio and Visual Studio Code. With Windows you can use Java for cross-platform application or C++/# for native ones. We can use node.js and REST for authentication. Using JavaScript with React would be a good option for cross-platform as well. These tools are free to use, but production licensing is around $138 - $628 per month. | For programming for multiple mobile devices, I would recommend Visual Studio Code. This will be used to code your base and use Git or Homebrew or Yarn for managing packages. Using a universal language like JavaScript, HTML, and CSS would be best for front-end development. It wouldn’t hurt to have XCode, and Android Studio downloaded to take advantage of the built-in emulators for testing. React-native would be a great choice for cross-platform web development hosting. While development is fast and cheaper, you may be stuck paying multiple fees to publish on different mobile platforms. Testing browsers on mobile would not be optimal. Usually the UI is altered and scaled differently, and not all browsers are available on mobile. |

## 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 that the gaming room should use the Windows OS platform. It is widely used and compatible with most IDEs and browsers, so you shouldn’t have an issue finding the best environment for the job. It also has a very basic architecture that makes it easier to teach and learn.
2. **Operating Systems Architectures**: Windows architecture is great for application development. It employs a layered architecture with a user mode and kernel mode that makes it easy to control your environment and resources. The kernel mode controls halt user mode services from accessing any sensitive areas of the operating system. These features are accessible by specific user accounts. This adds an extra layer of security to managing servers and protecting access to data.
3. **Storage Management**: Windows is very compatible with multiple types of storage. I would recommend direct access storage (DASD). Using direct storage will render the fastest load times and better graphics, it will also free up the main RAM on the computer. This makes better cross-platform capability because it makes the game dependent on network connection and browser rather than hardware requirements. Most DASDs come with built-in functions that enable reading and writing capabilities. For scaling, I think a cloud storage provider would be good. They provide scalability, security, and functionality at a monthly rate. They also provide HUDs to manage and observe utilized memory. In an online game that has timed rounds, Draw It or Lose It would benefit from this. Windows comes with built-in storage management like Storage Sense and Windows Storage Manager to make managing disk space easier. It also includes APIs that you can implement into the program.
4. **Memory Management**: For this game, we need a library of pictures, users, and their information. There will need to be the creation of a library (database) to store and organize this information going in and out of the database. Server providers usually provide means of managing memory in the database. Windows also comes with great storage and memory management tools built-in as mentioned before. The application would need to securely access this information using the read/write function from the server.
5. **Distributed Systems and Networks**: Development for multiple OSs wouldn’t be too difficult, but testing would be necessary. This game must be able to connect different users on different browsers, so we must keep that in mind. To accomplish this, you need a strong network connection to the internet and storage services. The client-server distribution works great for this design because there are multiple users on the same server, making data flow between them easier. However, when the user count increases, more server space will be needed. Using cloud (virtual) storage makes this very simple to do by providing a simple increase when requested, avoiding any unwanted crashes.
6. **Security**: Windows comes with many security features like Windows Defender, but it wouldn’t hurt to ensure security with virus protectors. Having individual user accounts for developers on a secured server and assigned roles would increase security with development. As for the application, we would need to encapsulate and authenticate data to and from the server. Authentication and authorization for users and their data paired with input validation is necessary when using any web-based application.