

Draw It or Lose It Browser

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/24/2024 | Charlie Hart | Initial draft of design document |
| 1.1 | 04/14/2024 | Charlie Hart | Changes to design constraints, expansion on and completion of platform evaluations. |
| 1.2 | 04/17/2024 | Charlie Hart | Initial draft of recommendations section. |

## [Executive Summary](#_heading=h.35nkun2)

The client, The Gaming Room, has a functional Android app which they want to bring to a web-based platform. The object of the game is to work with your team to guess the subject of a drawing from a preset library, steadily rendered in increasing completion over the course of thirty seconds, before the time expires. They are seeking to optimize compatibility between different systems on which the game may be played.

## Requirements

Only one instance of the game may exist in memory at a given time. The game must be able to have one or more teams, each team having multiple players. Game names and team names must be unique, with the ability for users to check whether a team name is already in use when choosing one.

## [Design Constraints](#_heading=h.1ksv4uv)

The chosen platform must be both secure and capable of hosting a server efficiently. The implementation for the client-side app will preferably be as universal as possible so as to run on Mac, Linux, and Windows with as few platform-specific changes as possible, and it will need to be able to run in a browser Window on each of these three platforms.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

The entire program exists within the com.gamingroom package and directory. ProgramDriver contains Main, which uses SingletonTester to verify the integrity of the singleton pattern used to guarantee that only one instance of GameService exists at a time. The Main class is also responsible for any procedural operation of the program. GameService handles all of the game information through the use of its public member functions and by keeping references to Game objects in an array list. In a similar fashion, Game keeps an Array List of Team objects, and Team keeps an Array List of Player objects. GameService has a zero-to-many relationship with Game, which has its own zero-to-many relationship with Team, which has, in turn, a zero-to-many relationship with Player. This means that each of these successive classes is not required to have an attached object of the respective type but is able to have an unlimited number of them.The diagram indicates that these four classes are associated, but in practice, they have a compositional relationship, with no instance of a child class existing in the program without its superclass. Similarly, Entity is treated as an abstract class from which Game, Team, and Player all inherit.

**"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](#_heading=h.z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | With relatively low compatibility and a penchant for unique architectures make Mac a potentially-problematic choice for cross-platform compatibility. These features do provide enhanced security, though. Coupled with Mac’s Unix terminal, these features make Mac a better choice for a power user in this application than a typical user. | Linux is highly customizable, scalable, and controllable. A lightweight distro selection paired with decent hardware will likely allow Linux to outperform the other options for the cost, at least of operation; the OS itself will likely be free. Using Linux decreases the likelihood of unintentional dependence on proprietary libraries, thereby also increasing the chances of cross-platform communication without errors. | Windows has a plethora of options for power users but is hindered somewhat by unnecessary features. Being the most popular desktop/laptop operating system, it also suffers from being the largest malware target by far. Windows has an upfront licensing cost to install fresh but does not require a subscription or royalties thereafter. | Not well-suited for server hosting. Internet connections tend to be wireless, and, while portable, the hardware capabilities are limited, meaning that even if a mobile device had access to and control over great enough bandwidth to operate the server, it would still be a poor choice for processing. |
| **Client Side** | Mac development is likely to prove the most costly due to the specific expertise needed to develop for the platform. Use of Java may offset this due to the JVM’s portability allowing for fewer adjustments. This can be achieved by distributing the game to desktop as an embedded Java applet. | Daily users of desktop Linux are often tech-savvy professionals and enthusiasts and are less common than Windows users. Development will likely be comparable in cost and effort to Windows but offer a lesser ROI due to the narrow market. Again, a Java applet will make this venture most profitable due to portability. | Windows is the most popular traditional OS. Development resources are readily available, but multiple major versions of Windows are still in use today. Nonetheless, compatibility and portability are high. Windows can and does run Java applets in browsers. | As long as phones can run similar enough versions of Android, Java and Kotlin are common, simple, and portable. Apple is similarly portable to itself but requires a larger investment of resources and knowledge to access a much smaller potential market. Mobile devices generally do not do well running desktop-designed Java applets in browsers; it is rare that they function at all. |
| **Development Tools** | C++, Java, and Swift can be used to develop apps for Mac OSX, in reverse order of preferability. Both Java and C++ can be written in Eclipse, as can Swift, though XCode is much better suited to the task. | C++ and Java are excellent languages for Linux development, and both have a high compatibility with Eclipse. Visual Studio also works well for C++, but it is developed by Microsoft and requires a subscription. | C++ and Java are both strong choices for developing this sort of app for deployment to Windows, as is C#. Eclipse is generally best for Java, while Visual Studio is preferable for C++ and C#. | Java, Kotlin, and Swift are the most popular programming languages for developing on mobile. Android Studio and Eclipse are most popular for Android, while XCode and AppCode are popular for iOS. AppCode requires a subscription, while XCode is available for free on OSX. Both Android Studio and Eclipse are available for free. The App Store and Google Play both charge a service fee of 30% of revenue, although both have voluntary enrollment programs to reduce this to 15% if revenue is under one million dollars yearly. |

## Recommendations

1. **Operating Platform**: Windows likely be best suited to the development and adaptation of Draw it or Lose It to other platforms. Porting of the existing Kotlin and/or Java code to the web-based environment using these platforms will be facilitated by the ready availability of both closed-source third-party and FOSS software on these platforms.
2. **Operating Systems Architectures**: Windows uses a layered architecture based upon a central kernel, to which the higher levels of the operating system can be thought of both as an access hatch and an elaborate abstraction. The kernel has executive processes to handle virtual memory management, power, processing, and the like. The user does not interact with the kernel directly but through the lens of the user interface at the user layer. The kernel mode consists of multiple layers comprising everything from the high-level CPU operation and virtual memory management to the direct management of hardware devices like the MMU and other I/O. In this way, the kernel itself is an abstraction of the CPU to other processes, serving as a means to interface with the processor.
3. **Storage Management**: If The Gaming Room opts to handle all of its storage management in-house, the modern standard of NTFS is the best candidate for the file system due to its scalability, portability, and security. A hybrid flash storage system will allow for larger files like images to be accessed quickly while offering the cost-saving benefits of a hard disk drive system. Both user-generated information and hosting information should be sorted into a tree-structured directory to allow for organization and ease-of-access, as well as to minimize the impact of overlapping user-generated information between unique game sessions. If The Gaming Room opts instead to outsource some of its storage management, they should use Amazon Web Services’ S3 to handle delivery of high-volume information to the player.
4. **Memory Management**: Memory paging and dynamic virtual memory management make Windows a promising operating system for multitasking and handling high volumes of unique information. Paging allows the kernel to efficiently free up volatile memory space by using the memory management unit to map physical memory locations to virtual memory, such that the memory is able to fulfill requests to a physical memory address when queried by retrieving the data from the associated page frame. Windows’ ability to do this dynamically makes it an ideal candidate for dealing with the number of large files used in *Draw It or Lose It*. The ideal use case of this functionality in the context of *Draw It or Lose It* is to position the user’s machine to be prepared to access the files it needs soonest from the virtual memory dynamically by feeding each subsequent file into virtual memory after the previous is loaded into physical memory. This design choice may result in excessive swapping and bog down the processor, though; in that case, it will be best either to tolerate the disk loading time or to make an upfront investment of memory at the start of each round by loading all required images at once.
5. **Distributed Systems and Networks**: The heart of distributed systems is in designing one or multiple programs which are capable of communicating betweens different running environments in order to make decisions based on external information. By sending packets of information between the users and the computer(s) operated by the gaming room, a client-server relationship can be used to facilitate the operation of *Draw It or Lose It*. Depending on the scale and required computing power, it is worth considering running the entire program in a cloud-based environment, as this simplifies a number of issues The Gaming Room may otherwise need to pay more money to solve in an inferior manner. AWS has servers around the world which can deploy needed assets to users and minimize the handling by the main game server of tasks unrelated to the main operation of the game, effectively implementing the beginnings of a microservices design philosophy should The Gaming Room wish to expand. This comes with the added benefit of outage resistance, as the redundancies of the AWS network ensure that when one server goes down, another is likely available to take its place..
6. **Security**: Windows has a built-in firewall system intended to stop common threats, and the NTFS file system offers access control and permissions. Additionally, secure protocols can be used to prevent unverified access to sensitive information by malicious actors. Perhaps most importantly for this application, the Java Virtual Machine has very limited ability to modify memory, only being able to reference objects, meaning that attacks by the avenue of the game itself will be of limited harm inherently. Additionally, a strong recommendation goes out The Gaming Room to take advantage of account management features offered by cloud providers like AWS and Google. Google is particularly useful, as it offers single sign-on integration with Google accounts. While this does require a Google account, it offers world-class account and sign-on security for both users and administrators, and granting role-based permissions based on correspondence from the SSO API will mitigate unauthorized access.