

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/14/2022 | Maxwell Marrone | Added information on executive summary, potential design constraints, and a domain model review |
| 2.0 | 9/30/2022 | Maxwell Marrone | Discuss server and client side requirements for various platforms, and development tools. |
| 3.0 | 10/13/2022 | Maxwell Marrone | Made formal recommendations for architecture |

**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 goal of this endeavor is to adapt the Android App for Draw It or Lose It into a web-based game for The Gaming Room. This project can be deployed on multiple platforms. The game will have one more team, each with multiple players assigned to it. Each team, instance, or player will have unique instances to accommodate only one instance of the game in memory. Similarly, game and team names will be unique.

## [Design Constraints](#_2et92p0)

As a web-based application, this project will need to be written or compiled in a way that is web friendly, this may have an impact on what tools can be used to bring this to life.

The actual service will need to be hosted somewhere so players can connect and join teams remotely, the server to do so will need to be determined. While The Gaming Room has not specified business requirements on budget, identifying maximum potential usage and how much they can spend on hosting will be important.

## [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 provided UML diagram, each GameService can have 0 or more Games, which can have 0 or more Teams, and 0 or more Players. ProgramDriver calls SingletonTester to verify there is only one instance of GameService running. Game, Team, and Player inherit from Entity for common attributes. Because they extend from Entity & Object, they are also polymorphic. Sensitive and private information is encapsulated within each respective class. The only way to retrieve values from a class is through abstraction.

**"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** | Mac environment comes backed with impressive hardware, and proprietary code that makes it harder to break into. Hosting may be more expensive overall. This might be worth it for the extra security and high quality expected performance. | Linux’s open source nature lends itself to hosting in several ways. It will likely be a cheaper solution, and may not need as powerful hardware to match performance on other platforms. Additional technical experience may be required to utilize Linux command line. | The regular and necessary updates to Window’s platforms to patch bugs and address new features may lead to the highest down time across viable server hosting environments. Some Microsoft technologies like .NET and MSSQL, if required for deployment, will make Windows hosting a necessity. | A Mobile device is not likely to provide a quality hosting environment. Even if one were to jailbreak / root their device and somehow download the tools to be able to turn a device into a usable server, the limited hardware and battery dependency would surely invalidate any benefits from using the system. |
| **Client Side** | Supporting clients on Mac may be more expensive than others, because of the necessity to obtain a Mac if not available. There is also specific expertise to consider: the Safari browser that only runs on iOS. That will be the main factor to consider regarding time. | Linux has the potential to be the most time consuming environment for verifying compatibility. Many versions of Linux exist including Ubuntu and Debian, and each has the potential to have multiple supported versions running at once. One server can be used to test different environments, and the hardware required is likely to be cheap. | There are several supported versions of Windows, and many browsers available on the platform. This will likely be middle of the road in cost, time, and expertise due to the relatively low cost (comparatively) to run and license a Windows server. It's high popularity will likely lend itself to having information available for any specific issues that may arise. | Mobile devices are likely the most expensive, and one of the most time consuming due to the large number of devices available on the market. Between iOS and Android alone there are plenty of manufacturers and distributors. Specific app expertise will be required here since it will likely not be distributed on a browser compared to the others. The other expenses related will be the app developer licenses needed for hosting on the stores. |
| **Development Tools** | HTML/CSS/JavaScript/PHP to display the webpage information, and communicate with the server. Java will be used for the backend.  Atom is a popular, reliable, and free IDE for Mac, and some Mac specific Apps exist like Xcode and Homebrew can assist in development. | HTML/CSS/JavaScript/PHP to display the webpage information, and communicate with the server. Java will be used for the backend.  Linux has a multitude of available IDE like Atom, Visual Studio Code, and Brackets. | HTML/CSS/JavaScript/PHP to display the webpage information, and communicate with the server. Java will be used for the backend.  Windows has a high number of IDEs available including Atom, Visual Studio Code, and Notepad++. | Java will still serve as the back end for mobile development. For iOS, Swift can be used within Xcode. For Android, Android Studio can be used as an IDE.  Development tools and resources like React or Xamarin can assist in development as well, serving as frameworks for apps. Additional fees to consider for deployment on app store, and due to the different deployment may require a different team. |

## 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**: The recommendation is to use a Linux based server for the app deployment. By not opting for a cloud solution, The Gaming Room will have a great deal of control over their own server specs, and will not be bound to many of the costs associated with cloud. On top of it, the server will not be limited by downtime of the cloud itself.
2. **Operating Systems Architectures**: The Linux operating platform begins at the hardware layer, which comprises all of the physical components that will allow the server to run. The system RAM, storage, GPU, and more are comprised here. Linux’s kernel layer then communicates between the hardware and the OS itself, which controls the basic functions of the platform. Finally, the application layer is where the code and execution of the app will live.
3. **Storage Management**: Storage on Linux is handled by one of any number of file systems. One of the most popular, XFS, can support a large file system and is the recommended one given the need to have and maintain a large database of information and users.
4. **Memory Management**: Memory is handled on the operating system with paging, and access to virtual memory as well as physical RAM. For The Gaming Room, it will be necessary to keep the images loaded into memory at all times to ensure fast access for the client. The main concern will be identifying how many total users can use the app at one time. With paging, total memory can exceed physical, and allow for more active users.
5. **Distributed Systems and Networks**: Due to the requirement that this game will be hosted by a server and distributed as a web app, the easiest way to accomplish this is via REST API communication. A standard API will allow apps run from browsers, or downloaded from an app store, to communicate effectively in the exact same way back to the central server. This method requires connection either over the internet, or cellular traffic to reach properly. This goes both ways, if the server loses its internet connection, or otherwise suffers a power outage, the game will not be accessible. But it is still the most efficient way to manage the requirement.
6. **Security**: Security for both The Gaming Room and end users is crucial for deployment. Many precautions can be taken to ensure client and server behavior is secure. Highly sensitive information, such as user passwords, can be run through encryption methods that cannot be reverse-engineered. The server will utilize the object oriented programming principle abstraction in order to ensure that all information available is relevant, and anything else is properly hidden. Similarly, least privilege can be assigned on the server itself so no more information than the bare minimum is provided to any who need it. In these ways, the client and server can remain secure.