

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

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

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

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

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

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

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

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

[**Recommendations**](#_heading=h.3rdcrjn)5

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

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/19/2022 | Cory Brandes | Initial design capture |
| 1.1 | 06/03/2022 | C. Brandes | added evaluation section data |
| 1.2 | 06/15/2022 | C. Brandes | Completed recommendations |

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

Create a web-based version of the game Draw It or Lose It. The software language and deployment environment is still unknown. The client will need to understand if cloud or local hosting is preferred. Will the target be mobile, pc or both. Are the use of open source libraries OK with the design. Each piece of the design will impact another. Setting a good backlog will be key to delivering on time with all the features needed.

## [Design Constraints](#_heading=h.tyjcwt)

A successful project starts with a good understanding of the deliverable. This begins with a design capture and creating a backlog of requirements. The language choice for development could be driven off the deployment Os. The web app will require extra development time to automate the environment setup for the client. Simplifying this process will ease the pain for the initial release and future updates.

## [System Architecture View](#_heading=h.3dy6vkm)

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.1t3h5sf)

The design is set up using the main class (GameService), the base class (Entity) and three child classes of Game, Team, and Player. The GameService class creates an object of the Game. The Game class holds a list of Team objects. The Team class holds a list of Player objects. The GameService needs an object of the Game to create a team. The Team doesn’t exist without an object of the Game. The Player can’t exist without a team.

**"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.2s8eyo1)

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** | Proprietary OS code base, security is priority.  Hardware and OS coupled to run efficiently.  Multiple login levels  Applications stored in user level folders.  Weakness:  OS coupled to specific hardware.  Expensive hardware solution. | Deployable to multiple processor types.  Hardware independent with open source drivers.  OS variations are security focused.  Weakness:  Open source kernel and driver code base. Opens up the possibility of security holes.  Inexperienced user setup running as root is bad news. | Hardware independent - mostly.  User login levels  Most applications are easily built for Windows.  Business and Home use widely adapted  Weakness:  Most widely attacked OS.  Registry and startup manipulation. | Development is done with a specific tool set for Android and Mac OS.  Licensing involved for application release.  Applications need to pass security checks before hitting the app stores.  Apple has minimal hardware variations.  Android has many hardware variations depending on manufacturer |
| **Client Side**  With the option for cloud based build and testing, you could technically develop for each platform if a common language was used. Most of the issues listed could be dumped under each OS. | Development license is needed to release software to the app store.  Multiple browser and screen resolution support needed.  Licensing is dependent on the language used.  Hardware is needed to develop and compile the code base or a cloud based virtual machine. | Linux environment knowledge is beneficial.  Cross platform IDE and project setup can minimize development start up time.  Multiple build configurations needed for testing on different virtual machines. | Windows could be used to develop and build for both Linux and Windows deployment.  Deciding which browsers are covered for development. | Knowledge of the development platforms is a must.  Code reuse between both Android and Mac OS will be minimal.  Deployment and testing on the different platforms will have an associated cost. |
| **Development Tools** | Swift, Java, C++, Angular, Python and C could all be used for development. Swift being the only language locked to a Mac device for design. The others can be developed with any IDE. Possibilities are endless. | Java, C++, Angular, Python and C could all be used for development. Setting up the environment will depend on the language and IDE.  Possibilities are endless. | Java, C++, Angular, Python and C could all be used for development. Setting up the environment will depend on the language and IDE.  Possibilities are endless. | Mobile has the native dev tools and multiple other options like Xamarin which offers cross platform development for most platforms.  Again, the options are wide open for most developers. |

## 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**: AWS Linux Cloud, Expandable with multiple instances
2. **Operating Systems Architectures**: Ubuntu 20.04 LTS X86\_64 allows local development prior to deploying to the cloud. Using the LTS version keeps security and updates available over the next few years. No OS is immune to cyber attacks but Linux has a good track record of fending off exploits.
3. **Storage Management**: The images could be stored in a single database with each server instance pulling from that database. If the image fetch becomes an issue, these could be moved to a location on each server.

The individual user data will be stored on each server instance.

1. **Memory Management**: Memory management like the storage can be tailored per instance of the server. As the user base increases, the memory can be increased to handle the workload. This includes increasing RAM or swap space to help smooth out the data movement.
2. **Distributed Systems and Networks**: The server can live in the cloud or as a local instance. The cloud-based approach can be expanded with multiple instances quickly to support the ramp-up of game play. The local server will have some hardware cost up front and more if you expand this option. The frontend application will be developed with native tools or the all-in-one options like QT and Xamarin. Interfacing with the server using a secure API will make the application expandable to all platforms.
3. **Security**: Security will be handled on the device and server level. Each platform frontend code will need to implement standard login practice along with verification with the server. Local user data will be stored in user space on the device, possibly encrypted or some form of translation needed to make it human readable. The server will only be accessed from an API that only allows read and write to specific items. Data transmission again will be encrypted in some form.