

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.4

## 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 | 01/22/2023 | Derek Bamford | Initial Design Document |
| 1.1 | 01/27/2023 | Derek Bamford | Updated Design Constraints |
| 1.2 | 02/05/2023 | Derek Bamford | Completed Evaluation |
| 1.3 | 02/13/2023 | Derek Bamford | Updated Evaluation |
| 1.4 | 02/19/2023 | Derek Bamford | Recommendations |

**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 Gaming Room, heretofore referred to as “the client” is looking to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It, which itself is based on the 1980’s television gameshow *Win, Lose, or Draw*. The client’s game is currently only available as an Android app. This design document will cover multiple facets of the development of the software and should be kept confidential and not presented to any entities outside of CTS and the client.

## Requirements

* Game must run on multiple platforms, including Windows, Mac, Linux, and other mobile platforms.
* Each game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

* Security Issues
* Game and team names must be unique.
* Web server to host the web application.
* Storge of all associated images
* How quickly can images be rendered across various platforms.

## [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)

The Entity class is the base class of the program and has the attributes Id and name set to private. The methods are public and call the private entity constructor and obtain the stored ID and Name using the assessor “get”.

Three other classes inherit the attributes and methods from the Entity class these are the Game, Team, and Player classes. The Game class allows for a new game and team to be added, the team class can add players to the team, and the Player class stores the Id and the name of the player. The game class has an association with the Team class which has an association with the Player class. The multiplicity for all three is set from zero to many instances. The Game Service class has an association with the Game class. It is in this class that we add and get game instances as well as get player and team Id information, this class has many private and public attributes and methods, which is an example of the OOP principal encapsulation.

Finally, the Program Driver class has a directional association that uses the Singleton tester class which makes sure only a single instance of a game can run at any given time.

**"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 OS X Server is UNIX-based and provides a stable platform for deploying applications, services, and technologies. Unlimited License is $999  Pro:  Easy creation of features. Lower Total cost of ownership (TCO)    Con:  Bad for scaling. High entry cost. Software compatibility issues because of closed system  Recommendation:  Use only if you do not expect high volumes of users | Deployment to Linux is officially supported for .NET Servers, using either .NET Core or the open-source Mono runtime.  No License Fee but servers cost $250-$1300/year to run.  Pro:  Low infrastructure requirements. Customizable system  Con:  No single official version of Linux. ASP.NET server upkeep is expensive.  Recommendation:  Use only if you have someone on staff that knows Linux | Has Windows Deployment Services (WDS)  $1069 for a standard Windows server Client Access License (CLA). $6,155 for a Datacenter CLA  Pros:  Great third-party support, intuitive GUI  Con:  High operating costs, more user-based licensing, and more virus security threats  Recommendation:  Most direct path to launch, but comes at a price | A mobile device can only be used as a server in a very limited capacity.  Hardware configurations vary from phone to phone.  Pro:  None  Con:  Not built to be a server for thousands of devices.  CPU and RAM limits and the need to be rebooted for updates.  Security issues  Recommendation:  Would not recommend they be used server-side. |
| **Client Side** | ASWeb Authentication Session to authenticate a user through a web service, including one run by a third party. The game should work on Apple versions of the most common web browsers  Contains built in network security features on devices.  Mac computers come at a premium cost.  Apple has a smaller market share.  Pro:  Easy to use GUI.  Con:  High entry cost  Recommendation  Would not recommend for smaller companies due to costs associated with hardware. More developers are switching from MacOS to Linux than ever before. | You can Authorize inbound traffic for Linux instances using SSH over IPv4.  Linux is a multi-user OS  ClamAV multiplatform engine for detecting trojans, viruses, and malware.  The game should work on Linux versions of the most common web browsers.  Pro:  Open-source applications  Con:  Very Low penetration among consumers  Recommendation  Community size increasing among developers, but consumer penetration remains very low. Ramp-up time depends on in house Linux skill set | The Microsoft Authentication Library for secure interactions between client and server.  Offers multi-user applications.  Highest market share among consumers.  Low cost of entry  The game should work on Windows versions of the most common web browsers.  Pros:  Easy to use,  high customer  penetration.  Con:  Resource hog  Recommendation  Would recommend using windows-based tools for the client side due to cheaper hardware, increased consumer penetration, and multi-user offerings. | XmlHttpRequest Javascript API or Web Sockets for server communications.  Cellphones have a very different UI than desktops, this will need to be taken into consideration.  The game should work on web browsers for both IOS and Android  Pro:  High market share for  smartphones.  Con:  No unified hardware  configuration  Not meant for multi  user functions  Recommendation  Would not recommend for use of client-side if also using other operating systems as hosts. |
| **Development Tools** | IDE:  VS Pro ($1200)  Xcode ($14.99/month)  CLion ($9-20)  Language:  Swift  .Net Maui  Framework:  Electron  Tauri  Webhosting:  XAMPP (Fee for commercial Use)  Recommendation:  Electron is Chromium with Node.js, which is the most popular server-side JavaScript engine. Both projects are well maintained and optimized. | IDE:  VS Pro ($1200)  Sublime Text ($80)  Atom (Free)  Language:  C/C++  Python  GTK is an open-source toolkit for creating graphical user interfaces.  Webhosting:  XAMPP (Fee for commercial Use)    LAMP ($9/month)  Recommendation:  A cheaper barrier of entry, but GTK has a steep learning curve. | IDE:  VS Pro ($1200)  IntelliJ ($15-50)  Xcode ($14.99/month)  Language:  JavaScript  Swift  Framework:  Phaser.js (2D)  Babylon.js (3D)  Cloud Computing:  Azure (Pricing varies)  Webhosting:  XAMPP (Fee for commercial Use)  IIS (License required)  Recommendation:  The option for updating to 3D from 2D pictures in the future could bring in additional users. | IOS:  IDE-  Xcode ($14.99/month)    Language- Swift  Android:  IDE – Android Studio  Language- Java or  Kotlin  Backends: AWS Amplify or Firebase  Cross-platform:  Flutter->Dart  React Native->JavaScript.  .Net Core-> C#  Recommendation:  The cross-platform pipeline would eliminate the need for multiple teams |
|  |  |  |  |  |

## 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**: A server OS has higher levels of functionality than a desktop OS, in that it can manage security, resources, and back-end functionality. A server OS also allows for unlimited user connections, greater memory, and can act as a web, database, and email server simultaneously. The latest data suggests that 80% of all server OSs being used are Linux based, which is why I am suggesting using Ubuntu for the server OS. While not specifically a distribution server, software will need to be installed, it is very user-friendly, offers regular security updates, and can scale to meet the future needs of Creative Technology Solutions.
2. **Operating Systems Architectures**: As far as hardware is concerned the Ubuntu OS is officially compatible with the three most common processor architectures-x86, AMD64, and PowerPC. Ubuntu uses the Linux kernel, which is the core interface between the computer’s hardware and its processor. Ubuntu also can use command line utilities such as *curl* to interact with RESTful API requests to transfer data to and from a remote server. Syncthing is a feature that exposes a REST interface over HTTP on a graphical user interface(GUI), allowing you to synchronize your files across multiple devices. This means the creation, modification, or deletion of files on one machine will automatically be replicated to your other devices, which will be beneficial for the next topic of storage management.
3. **Storage Management**: I would recommend a cloud database, such as AWS, Oracle, or Azure to store all the game information including game images, username, team name, win/lose ratio, and any other metadata needed. The advantages of using these databases are less hardware dependence, enhanced scalability, and security. The only images stored client side would be those that are specific to the current instance of the game being played.
4. **Memory Management**: The Linux kernel divides memory into three memory zones DMA, Normal, and Movable. Memory below 2 GB and some I/O operations are placed in the DMA zone. Memory above 2 GB can be used for all memory allocations and is placed in the Normal zone. Finally, the Movable zone can only be used for memory buffers that can easily be moved by the kernel. Memory in this zone can more easily be taken offline than memory in other zones. The way Linux handles loading executable images into a process’s virtual memory is through demand paging. Demand paging is when the operating system copies a disk page into physical memory only if an attempt is made to access it and that page is not already in memory. For Win it or lose it, the images will be stored server-side and cached on the individual’s device when loading the game through the browser, once the game is closed the memory will be wiped.
5. **Distributed Systems and Networks**: The most common way a client connects with a server is through the TCP/IP protocol suite. The TCP allows for connections to remain established and deliver packets of data until the connection is stopped. IP, however, is a connectionless protocol, meaning each packet remains independent of any other data being exchanged. A RESTful API is like the TCP/IP protocol for applications in that it also determines how applications interact and exchange data with each other. REST APIs are stateless, and communication can only occur one way when a client makes a request to the server. All REST APIs follow the same uniform interface guidelines for communication and are layered systems meaning there are more servers between the client and the actual server they are communicating with which helps with security, data traffic, and other functions. Rest API also can cache data to a client’s device for later use, to help speed up load times.
6. **Security**: Operating systems prevent users from unauthorized access to the resources defined by a computer system in a variety of ways. One of the most basic principles is the principle of least privilege, it is the idea that one gives users only the privileges necessary to perform their intended tasks, in *Win It Lose It’s* case playing the game on a web browser.  This principle also has the added benefit of compartmentalizing the systems so that if a component is compromised or fails it will result in the least amount of damage to the underlying data or systems.  The best way for the client side to be secure is for a simple password log-in system, but if you want a much more secure system a two-step authorization process is even better.