

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 | 03/19/21 | Andrew Eskild Christensen | Filled out details regarding project requirements and criteria. |
| 2.0 | 04/05/21 | Andrew Eskild Christensen | Fill out Evaluation details. |

**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)

Creative Technology Solutions would like to create a web-based version of their Android game Draw It or Lose It. The game will fill out a drawing over the course of 30 seconds and the current team tries to guess what is being drawn. If they fail to answer within the time limit, the other teams get 15 seconds to guess what the image is.

## [Design Constraints](#_2et92p0)

* Game must be implemented using web-based technologies
* Game supports ability to have one or more teams each with multiple players
* Game and Team names must be unique
* Game should only allow one instance to exist in memory

## [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 entry point for the game is the ProgramDriver Class. This class contains one function, the main function. This class then uses the SingletonTester class, which has one function: the testSingleton function. Next, there is a singleton class, GameService. It has several private variables: games, nextGameId, nextPlayerId, nextTeamId, and service. Two of these have public getter functions: nextPlayerId and nextTeamId. In addition, GameService has getInstance, a function implemented as part of the singleton design pattern, which gets the singleton instance of GameService. The constructor is private, so this is the only way to get an instance of GameService. There are also a few functions to access the games list. These include addGame and two variations of getGame. Lastly, there is also a function that returns the current number of games. Speaking of games, the GameService class may contain zero or more games of the Game class. Each Game object has a private teams variable along with a constructor, a function called addTeam, and a toString function. Game, along with two more classes Team and Player, inherit from the Base class Entity. The Entity class has two private variables, id and name both of which have public getter functions within the class. The default constructor for Entity is private, and it has a public constructor that takes in two parameters, id and name. It also has a toString function. Because they inherit from Entity, the Game, Team, and Player classes also have the previously mentioned functions and variables. The last two classes are Team and Player. Team has a private variable, players, and two functions along with a constructor. The two functions are addPlayer and toString. The Player class is fairly simple, with only a constructor and toString function.

****

## [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 is capable of performing as a host for web-based software, however it really isn’t the best choice for that. A product called Mac OS X Server was shipped alongside Mac OS X as a separate operating system with server capabilities built in. However, over the years Apple has significantly reduced the functionality of the operating system to the point where it wouldn’t really be functional without a lot of extra work. They deprecated many of the services from the Mac OS X Server product itself, including DHCP, DNS, and VPN services. | Linux has a history of strong performance as a host for web-based software. There are many different solutions and a lot of experience in the industry on using it for that purpose. There are several distributions of Linux that come prepackaged with server software, though most of the software is open-source and can be installed on any distribution. One of the main hurdles to utilizing Linux as a server is the level of experience and time required to set it up, as a lot of things would need to be set up manually. On the plus side, because Linux is open-source, it is possible to set up a server with no additional costs for licensing, etc. | Windows, in the form of Windows Server, is a good option for hosting web-based software. The only downside is the cost that comes along with it in terms of license fees. Because the game is intended to be used by players throughout the world via internet, the Standard Edition of Windows Server would be the ideal choice, coming at around $972, but depending on the features needed that price would likely increase. The greatest reason to choose Windows Server is the ease of use and set up, which in itself could save enough money to make it worth the fees. | Attempting to use a mobile device as a web host would likely be a very impractical and inefficient option. There aren’t really any solutions readily available for that use case, and the majority of mobile devices lack the required processing power. |
| **Client Side** | Because the software is web-based, there is really very little issue associated with supporting Mac as a client. There may be some need to ensure compatibility with the Safari web browser, as that is the most widely used browser on that platform. | Most widely used web browsers available have support for the Linux operating system. As long as those browsers are supported, Linux support will come with it. | Just as with Linux, supporting Windows is mostly a matter of supporting the major web browsers: Chrome, Firefox, and Edge. The only difference is that many people on Windows still use Internet Explorer, so that will need to be taken into account. | The primary browsers on mobile devices (Chrome, Firefox, Safari) are based off of their desktop counterparts, so support will largely come with supporting those browsers. The smaller screen size and difference in input method will require some additional work to support touch screen and software keyboards for example. |
| **Development Tools** | Web-based technologies come with a lot of freedom in terms of the tools/IDE used to development them. Typically, in order to develop software that will run natively on a Mac or iOS device, you’d have to use a Mac with xcode but with web-based software that is not an issue. If the client wanted a native interface for accessing the game, there are many choices that allow for that while still using the same code-base as the web-based version, such as React Native. However, if the goal is for a fully native experience, then likely Swift would the language of choice. A final option would be a game engine, such as Unity or Unreal Engine. Each supports 2D game development and can be deployed on any of the major platforms. | This again is not an issue. There are a number of available tools and IDEs that can be used for developing web-based technologies. Many of the same technologies that allow for running a web-based application natively on Mac also support Linux. If a fully native version was desired, either GTK or Qt would be the best choice. Again, Unity and Unreal Engine both support deployment to Linux, so that is another option. | Windows is the most widely adopted operating system today. This means that not only is it an important target platform for any game, but it is also widely supported in making those games. The same technologies that allow for development of web-based technologies would be available here as well. In addition, for native development, Windows has the UWP framework. There are many ways that the framework can be used with different languages and frameworks, and it simply makes the experience more familiar and native to users of Windows. | There are really only two relevant platforms to consider in the mobile space and they are Android and iOS. Both are supported by Unity and Unreal Engine out of the box. Android applications can be written with Android Studio and the Android SDK for a more native experience. iOS development would not only require access to a specific SDK and IDE, but it would also require access to a Mac computer, as that is the only way write software for iOS if it is to be a native application. If going that route, you would need developers with experience in write Swift code. Both platforms are supported by React Native. |

## 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 would recommend the use of Linux as a host for the web-based application, allowing users to connect to the game from any available web-browser on their system. There may be greater work initially in setup, but as long as they can find a competent system administrator with experience in administrating Linux servers it will be more cost effective in the long run.
2. **Operating Systems Architectures**: Linux is the name used to refer to a variety of different operating systems all with one thing in common, that they all use the Linux kernel. The various operating systems are referred to as Linux distributions. Each has their own set of software and design philosophy. Because the Linux community believes strongly in open-source technology, most of the software you will find running on Linux is open-source and many distributions use different combinations of the most common open-source projects. Some of the more widely used distributions include Ubuntu, Fedora, and Manjaro. Manjaro is an Arch Linux-based distribution that has gained popularity in recent years, however its use of a rolling-release update cycle and lack of major financial backing make it a less stable options and is therefore not typically used in an enterprise setting. Ubuntu and Fedora are both well regarded and trusted options that both use the Gnome Desktop as default, though both have branching versions with other options for the software packages that come preinstalled, such as KDE. I would recommend using Ubuntu as it is the most widely supported and mainstream option. The primary components that will be used for the server operation are Apache, PHP, and mySQL. Apache is the HTTP server and it does the job of actually delivering the content from the server to the client machines when they request it. PHP is a server-side programming language that allows for creating more dynamic content. mySQL is a database management system that would be the method by which we would store user data and access it when needed.
3. **Storage Management**: The server will not only be storing data about games being played, but also sensitive user data such as usernames, passwords, etc. With this in mind, it is of the utmost importance that that data is handled carefully, and the proper steps are taken to keep it safe and intact. Periodical backups should be performed, and there are several options for that. Bacula is a well-known open-source project that can provide that functionality. All sensitive user data should also be encrypted, and only the absolute bare-minimum of user data should be stored.
4. **Memory Management**: Linux memory management is handled by the kernel. The memory management subsystem is responsible for this, and does so through the use of virtual memory, demand paging, memory allocation, and many other techniques. By limiting the number of instances of GameService that are allowed, we reduce the required amount of memory. In addition, by separating the backend software that will be handled by the server from the frontend software that will be rendered client side we offload the processing of the frontend to the client device, reducing system resource use as well. In order to help mitigate memory usage issues it might be helpful to implement a queuing system to help manage the server load by managing the amount of games being played at one time.
5. **Distributed Systems and Networks**: Allowing clients of all different platforms to communicate with the Draw It or Lose It software will be accomplished using an api, for example a REST api or a GraphQL api. It wouldn’t matter what platform the user was on, their browser would simply send commands in the form of these api calls and the server would receive them and respond accordingly. Because users will be connecting through a network, there may be instances where their connection is interrupted. This can me managed simply through a timeout. If the user doesn’t interact with the site within a certain period of time, say 2 minutes, they will be assumed as away from keyboard, or disconnected, and removed from the game. We can implement something similar on the client side to indicate to the user if they are having connectivity issues by measuring the time it takes from when a request is sent by the user to when the server responds to determine if they are having network issues and indicate that to the user.
6. **Security**: Protecting user privacy is of the utmost importance. User data will be encrypted when stored and encrypted when sent over the network using https/TLS. To protect our user accounts from unwanted access in the case that their password does get in the hands of someone it shouldn’t, we should implement some form of multi-factor authentication.