

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)4

[**Recommendations**](#_m8aleynsvzvc)7

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/21/2021 | Sam Brasher | Creation of Document; |
| 1.1 | 06/01/2021 | Sam Brasher | Addition of Evaluation section |
| 1.2 | 06/13/2021 | Sam Brasher | Addition of Recommendations section |

**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)

We will be designing an app based on The Gaming Room’s current game, Draw It or Lose It. The aim is to make this app web-based and available on a wider variety of platforms. We need to be sure only one instance of the game exists across platforms, likewise with team names.

## [Design Constraints](#_2et92p0)

We need to ensure that all processes and data are synced between our servers and users. This includes challenges like varying network connectivity, validation of secure connections, and the end-user environment. We will also need a compatible storage solution for the set of stock images.

## [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 this class diagram, we see that the Program Driver uses the Singleton Tester – in this instance, to ensure that only one instance of GameService exists. We can also see that the Game, Team and Player class inherit from the Entity class. Finally, we can see that the Game Service is associated with zero to many Games, Games are associated with zero to many Teams, and Teams are associated with zero to many Players.

****

## 

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macintosh is very rarely used for server-side development. However, Swift has recently been updated with server-side language. Swift is free to use, but we would build our framework from scratch. | Linux has a variety of free server-side languages. There are many helpful resources available. Linux is the most used operating platform for server development, which speaks to its usefulness. Resources like Apache are available. | Windows is another good option for our server-side development. A variety of free languages exist. Apache is also available on Windows. | Mobile devices would not be ideal for hosting a web-based software application. Apache is available on Android, but we are severely limiting our capabilities by hosting off a single mobile device. |
| **Client Side** | Developing an application with Swift would be relatively easy, with Apple providing a large of amount of support and free resources. Devices are constrained to a limited amount of known hardware, easing development concerns. Costs are related to purchasing hardware for development. | Developing an application for Linux is free and a test operating platform can be installed on a variety of hardware. There are many potential variations of Linux clients, which may add to development costs. | Developing an application for Windows is free. Windows can run on a variety of hardware, which means a variety of different potential configurations of the operating platform. This could add to development costs. | Developing for iOS or Android is free. iOS runs on limited hardware options, which can ease development costs. Android is run on a variety of different devices, presenting a myriad of potential configurations which could add to development costs. |
| **Development Tools** | Swift is swiftly becoming the language of choice for macOS applications. Access to the language is free, as are support and resources. Swift can be developed on a selection of free IDEs like XCode and Visual Studio. If we are not developing on Macs, we will need units for testing. | Many major languages can be used for application development on Linux. C/C++, Java and Python are well known options, which can be developed on many IDEs. If we are not developing on Linux, we will need units for testing. | Like Linux, many languages can be used for application development on Windows. Most IDEs will be able to code with the major languages involved (C/C++, Java, etc.). If we are not developing on Windows, we will need units for testing. | iOS will use Swift for development (built on Objective-C). Swift can be developed on a limited number of IDEs. Android can be developed and emulated on Windows, but it would be nice to have units for testing. |

## 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 recommend that our server runs in a Linux based operating system. Resources exist to assist in creating this server, and Linux has a storied history as a server-side operating platform. Each game client can poll the server for what it needs. We will not have to pay licensing fees for each instance of the operating system, which could cut costs, though there are several enterprise server platforms built on Linux that we may wish to use. A platform like Red Hat Linux offers support at a premium. A SQL server also runs on Linux, which could be a preferable option due to more established support.
2. **Operating Systems Architectures**: A distributed operating system will be tremendously beneficial to our design. Having redundant components separated by physical space ensures that our system maintains operation if any one part fails. We can use a major service like Amazon Web Services to host our equipment, saving us costs on maintaining our own servers.
3. **Storage Management**: We can use a SQL database on Linux to store our pictures. Likewise, we can use this to store a database of teams’ global rankings, a database of hashed username and passwords, and any other types of information we may need to store. Our database of pictures will not take up much storage space at 2 GB, but as our user base grows so too will our database of user information.
4. **Memory Management**: SQL servers like to use all available RAM to ensure the server is functioning at maximum capacity. If we would like to have the entire database of pictures loaded into memory, we should have at least 2 GB. Given that the operating system itself needs more RAM than this, we should shoot for a more reasonable 32 or 64 GB of RAM for our server. We need to ensure that no memory leaks occur due to poor programming practices.
5. **Distributed Systems and Networks**: Using a service like Amazon Web Services, Microsoft Azure, Oracle Cloud, or other distributed systems providers will help tremendously as far as ensuring our service stays operational. By relying on component redundancy, these services ensure that our clients can connect to our server despite any temporary outages. Complete outages, however, are not unheard of. See AWS’ or Fastly’s latest outages, causing several other websites to go down. We cannot 100% prevent outages from happening but going with a larger company could help ensure the service is back up and running sooner.
6. **Security**: We need to straddle the fine line between having fantastic security, and not being so overly secure that we drive away our user base due to inconvenience. A two-factor authentication method may be the solution, but this strikes me as something that may be only convenient on mobile platforms like iOS and Android. Using a salted hash method like Bcrypt, or a key derivation function like PBKDF2 would go a long way towards securing our user information. There are several recommended standards for implementing these algorithms, which we should consider and follow.