

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 | 05/22/2021 | Russell Pallas | Original draft of application and constraint solution propositions |

## [Executive Summary](#_sbfa50wo7nsh)

This application must allow for multiple users to play a game on multiple teams, however only one game is allowed at a time with a given name and ID. This constraint is enforced using a Singleton design, which will only permit one instance at a time to be created. Also, given this this will be a web based application, keeping memory and storage use to a minimum is a must.

## [Design Constraints](#_2et92p0)

Given that this is a web-based application, we would like to keep packet transmission to a minimum for multiple reasons. One is that not everyone has a very fast network connection and therefore if we can keep packet transmission to a minimum the user will have a better experience. We could combat this by having all images loaded on users’ devices prior to starting a game. These images could be loaded on a game-by-game basis so unnecessary images aren’t loaded if they aren’t going to be used in a game.

## [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)

This is a fairly simple UML diagram for this application. Game, Team, and Player all extend (inherit) from Entity. We also have bidirectional associations between GameService, Game, Team, and Player. ProgramDriver also has a dependency on SingletonTester. We are creating objects and placing them into vectors, which is a far better choice than a linked-list, for example. Speed in a game is a must and a linked-list does not create a fast experience for the user and results in far more processing on the back-end.

****

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * Most secure choice * Similar performance to Win and Lin. * Development time similar to Win and Lin. Shorter than mobile. * MacOS Server is a good choice for smaller projects * Significantly lower cost than Windows. * Unix | * Industry leader in server/network support. * Most universal choice * Security concerns given universality and ease of access to Kali. * Similar to MacOS server cost. | * AWS integration is a great advantage. * Can be less secure than MAC in some regards, though performance should be similar. * Greater market perforation than mac, but less than Linux. * Leasable for a low cost. | * NOT recommended for hosting. * Security concerns * Lack of integration / perforation * More difficult to access roots when needed * Reliability is a major concern. |
| **Client Side** | * Less cloud-based integration possible; less compatibility across browsers. | * More cloud-based hosting options; more cross-browser compatibility | * More cloud-based hosting options; more cross-browser compatibility | * Less cloud-based hosting options; less cross-browser compatibility. * More difficult to install and manage extensions. |
| **Development Tools** | * Java development is possible on MAC (Eclipse, Kite, NetBeans, IntelliJ; Xcode does not currently easily support Java) * C++ also possibly choice given OOP requirements (not fully compatible with DropWizard). * Mac development may cause portability issues. * All free community IDEs, with optional paid versions. * Two teams required (desktop and mobile); higher cost. | * Similar IDE choices as Mac and Windows. * BlueJ IDE also available, integration with Ubuntu. * All free community IDEs, with optional paid versions. * One team required; lower cost. | * Same IDE choices as Mac. * Visual Studio also an option (high market perforation and useful extensions) * All free community IDEs and extensions, with optional paid versions. * One team required; lower cost. | * Windows based devices mostly used Java for mobile. * Mac based devices require Swift or Objective-C within Xcode. * All free community IDEs, with optional paid versions. * Multiple teams required; higher cost. |

## 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**:

Recommended operating platform would be without question be a UNIX like distribution. The market perforation alone is reason enough. Considering cost as a factor, we most certainly want a UNIX like platform. Also, a LINUX platform will allow for the most customization of all of the options to suit the client’s needs.

1. **Operating Systems Architectures**:

ARM architectures are certainly the way of the future. Again, considering cost, ARM devices are becoming cheaper to manufacture and purchase all the time. They also are becoming more prevalent in mobile devices., which will be a major market for this application. x86 is also a good choice and isn’t going away anytime soon, however, the way of the future (although ARM has been out for a while now). ARM allows for a greater number of cores as compared to x86 and the more threads a server is running is typically better. ARM is also likely to allow for greater access to popular ports. We should also consider the thermal aspect of ARM. ARM devices tend to run much cooler than any other architecture. The downside to going with ARM is the lack of market perforation and technical support as well as the presence of likely software issues. However, these issues tend to be worked out rather quickly when it comes to server support.

1. **Storage Management**:

Storage management is a little trickier than memory management when it comes to Linux specifically. There is no real difference when it comes to Linux over other platforms. All choices remain on even playing ground here. However, given Linux’s customizability, we can say that there will be a greater chance for us to minimize the storage aspect of the server itself. Meaning that with the right combination of ARM processor option as well as adequate amount of memory for the images and users, we can minimize the amount of storage needed for this application. However, one thing that we cannot hand off to memory necessarily is user keys, both player and admin alike. Since security is so important to CTS, we could require more complex user keys, however, Linux security systems are already so good that we should not need to implement these changes. As for the methods that we will use when it comes to storage, we must remember that most storage is SSD now. SSD is quick and reliable, but only for so long. If we are constantly reading from storage on the server over thousands of games played by tens of thousands of users, we should expect to see a decrease in SSD performance over time, as is typical with any SSD. Multithreading should help us out here greatly, however.

1. **Memory Management**:

When considering memory, we must remember the fact that this game is meant to be played rather quickly. Therefore, given the fact that we will have multiple requests for larger files (images at 8MB/image), we should allocate a greater amount of RAM for the server to be dedicated to all of these requests. However, we may consider allowing for grouping of images to be allocated to RAM from storage given the fact that there may potentially be hundreds of games occurring at the same time. Although there are only a couple hundred images, people will likely only play the game with a couple hundred images for so long. This number will need to be expanded. Given this expansion, there will be an even greater number of image requests every second. Linux is a great choice for this because it is so customizable. Especially if we include the fact that ARM processors can now (for a cost) be custom made to fir a client’s needs.

1. **Distributed Systems and Networks**:

Network inconsistencies between users is something that we have absolutely no control over. The most control over this that we would have is to require users to use a wired connection, however, in this age there are very few people who are still using wired connections. We could require that users have a maximum ping time to ensure seamless use of the application (to the best of our ability).

Serverside however is a different matter. There we have almost full control over network reliability. The greater the number of components and dependencies that we have for our service, the greater the chance of having some sort of issue arise, whether it is minor or catastrophic. Also, given that this is a smaller outfit, we do not necessarily have the budget to be fully outfitted like Google or Facebook. For these reasons, as well as multiple platform communication, Linux really outshines the others. Also worth noting is the fact that Linux, for the most part, is open source so cost should be less of an issue.

1. **Security**:

Linux is a great choice when it comes to security. The Linux kernel is probably the most secure out there. One way to ensure proper security is to use SELinux (Security Enhanced Linux), which essentially limits the resources that one particular service may attempt to access on a server. We may also implement TCP wrappers. I believe that the Fedora distribution uses SELinux by default. Also, Linux is very strict with its permissions and requests must typically be accompanied by proper permissions. We will also have an easier time using Kali Linux to initiate some penetration testing. However, this also means that those familiar with the OS will have an easier time finding some open port or another weakness.