

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.30j0zll)1

[**Table of Contents**](#_heading=h.1fob9te)2

[**Document Revision History**](#_heading=h.2et92p0)2

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

[**Design Constraints**](#_heading=h.3dy6vkm)3

[**System Architecture View**](#_heading=h.1t3h5sf)3

[**Domain Model**](#_heading=h.4d34og8)3

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

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

## [Document Revision History](#_heading=h.2et92p0)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 03/21/21 | Ryan Mitchell | Completion of Executive Summary, Design Constraints and Domain Model |
| 1.0 | 04/04/21 | Ryan Mitchell | Completion of Evaluation, rhetoric of first section still needs to be updated |
| 1.0 | 04/16/21 | Ryan Mitchell | Fixed evaluation table to be more specific and on topic |
| 1.0 | 04/18/21 | Ryan Mitchell | Completed the recommendation 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\*](#_heading=h.tyjcwt)

In accordance with our client company, Gaming Room, we have been tasked with developing a basic environment to facilitate the development of their web-based Android gaming app. In addition to this, our client has requested that our program have the ability to involve multiple teams with multiple players, check for unique names of both games and teams, and has to run with only a single instance of the game in memory at any given time. In order to uphold this, we will be using a Singleton game service class that prevents creation of simultaneous game services, and an iterator to check our naming array for any instances of existing names upon creation. The service will be very basic in its construction, but will allow for continuous development.

## [Design Constraints\*](#_heading=h.3dy6vkm)

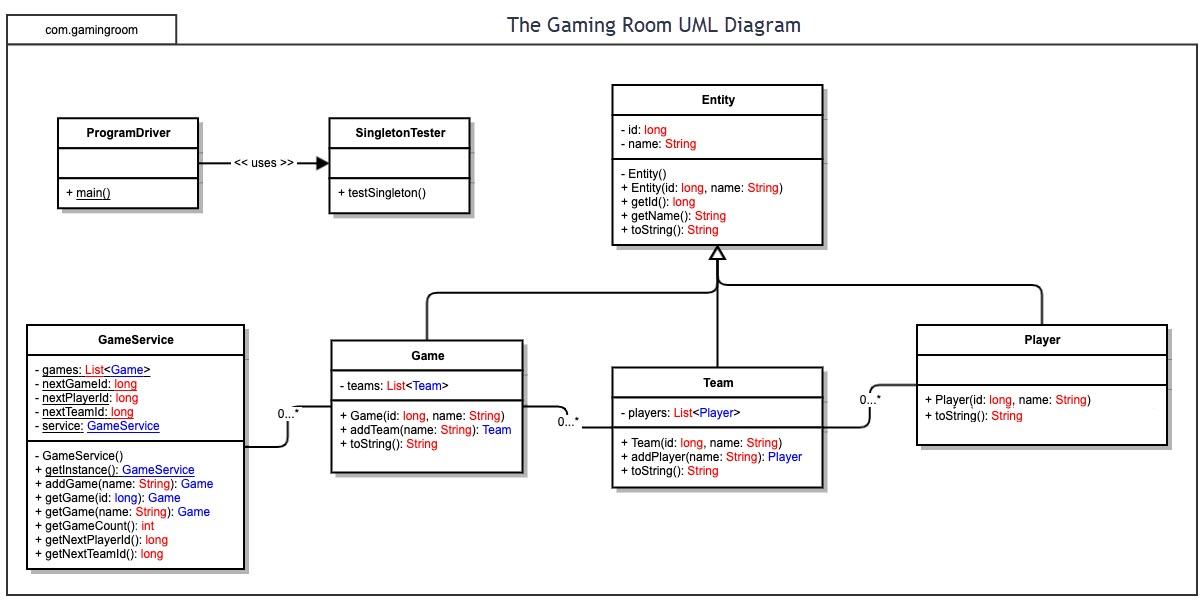
The program itself will be created with fundamental java techniques and will be accompanied by some sort of graphic interface. Our team will be using tools and software strictly within the Windows OS environment as this is what is both available and what our developers are most comfortable with. The environment will first be designed without functioning gaming applications and will require much more future testing to ensure bugs are handled appropriately when implementing games.

## [System Architecture View\*](#_heading=h.1t3h5sf)

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.4d34og8)

Within our provided UML class diagram, it will first be evident of the use of a ProgramDriver class to hold our main() method. This method has a unidirectional association with our SingletonTester class in order to demonstrate that only a single instance of our game service is running at any time through a testSingleton() method. Separate from this relationship, we establish an Entity class to create objects of basic attributes such as id and name with constructors and getters. Inheriting this class are several classes of different objects including Game, Team, and Player which all have a 0 to many association with each other. Each object class possesses unique identifiers for pertinent information such as constructors, addNew() methods, and toString() methods. The game class will contain a list of teams, and the team class will include a list of players to make sure information is correctly stored and recurring names are prevented. Connected to our Game class will be the GameService class, of which only a single instance can be run. The game service object will be able to retrieve information and start games by locating games based on the name or id that is passed to it. It will be able to iterate through teams and players to display their information if needed, and will store games that are created separately.

****

## [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** | If you can get past the exorbitant price associated with the Apple branding, a MacOS server would not only be easy to use, but extremely secure thanks to Apple’s high level of security. A Mac hosted server would be especially useful if the development team is mostly proficient with MacOS. | Unlike the MacOS, Linux hosted servers are on the opposite end of the spectrum in terms of cost and manageability. Due to the open source nature of Linux, server licenses are unnecessary, and the nature of open source software using fewer resources help make Linux a very efficient system. What it loses in terms of user friendliness it makes up for in it’s incredible versatility as far as configuration and security. | Somewhere in the middle, you have the very popular and polished Windows server. While costs are much more easily accrued, Windows servers and applications are generally well designed and supported for easy integration. The size and popularity of Windows offers it incredible support in terms of community and troubleshooting, and even offers remote desktop access through its GUI. This tool is great if you’re also designing software with the .NET framework. | While mobility can be useful, the lack of processing power and limited UI would make a mobile device a poor choice of server host. |
| **Client Side** | For our web hosted application, MacOS offers compatibility with browsers such as Chrome, Firefox, and Apple’s own Safari. While Chrome boasts great speeds, and Firefox promotes extensions and adaptability, Safari’s own unique ability of integrating with iOS functions gives it great security and even iPhone compatibility. | With the open source, DIY nature of Linux, there are few browsers unsupported by this OS (of which Safari is one). Chrome, Firefox, Opera, as well as many smaller name web browsers are used, which implies development for Linux on the client side would be straightforward, but may be limited in it’s mobile compatibility. | Windows browsers are largely the same in terms of availability as Mac, with the exemption of Safari being MacOS specific. This is replaced by Windows’ very own Microsoft Edge however which was introduced in Windows 10. Development compatibility is more easily narrowed, and with an already functional Android application, makes Windows a logical contender for focused client side development. | With the two primary smartphones on the market being Android and iPhone, these devices should be considered under their respective OS administrator. Android/Windows and iPhone/MacOS are identical in terms of browser application support, but an independent mobile application would require greatly stretched resources and almost a whole new branch of development. |
| **Development Tools** | Mac does support IDE’s such as Eclipse, Xcode, Visual Studio, Sublime, among many others. Mac also offers a virtual environment for the creation and testing of applications on other operating systems, however the price of both software license costs and hardware will be significant given the nature of Apple’s business model. However, Mac tools are generally well developed and easy to use for even a beginner Mac user. | The open source nature of Linux means that development tools will be mostly free. However, these tools are less polished and come with less bells and whistles than it’s primary competitors, which means more time and effort must be spent configuring and developing. It will also require expertise on the part of the development team to effectively develop on a Linux environment. | Windows is a place of comfort for many PC users, and while it is comparable to MacOS development in terms of software licensing, Windows is known for its ease of use and flexibility. There are numerous service and open source development tools to be chosen from, making Windows a great middle ground option in terms of tool availability and moderate cost efficiency. | Development of a large scale web hosted gaming application on a mobile device is simply inefficient. While it provides developers options as far as location, the development process would simply take too much time. Mobile devices are also limited in terms of memory and storage, making it the least favorable option of all four potential operating systems. |

## 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**: Given the already developed Android support, I believe using a Windows server as an operating platform is a great place to start. Windows has always kind of been the jack of all trades, providing great options in terms of licensed software and open source integration. While not as comparably strong in certain areas of specialty for other operating platforms, such as security on MacOS, it’s popular household reputation and remote accessibility make it a great option, even if for a moderate price.
2. **Operating Systems Architectures**: Within the Windows operating system architecture is a two part layered design that provides two major control functions which are user mode and kernel mode. As part of the user mode, there are also several different environment subsystems with their own API sets that allow for greater range of application integration, including Win32, OS/2, and POSIX . This gives users a lot more software availability for different tasks. Windows also provides a Kernel mode, which gives access to executive services which are not usually available in user mode. While giving access to functions such as I/O manager, object manager, cache controller, memory manager, etc., it also gives decent protection against access of the deeper, more critical functions within the operating system. Interfacing with the executive function in Kernel mode, there also exists Kernel mode drivers for hardware device interaction, and a microkernel which is limited to core functions such as first-level interrupt handling, thread scheduling, etc. The metaphorical protective barrier within the Kernel mode is the hardware abstraction layer, or HAL. It is the layer that exists between the physical hardware and the rest of the operating system. It contains hardware specific code that controls a lot of I/O interfaces with interrupt controllers and multiple processors.
3. **Storage Management**: With the price of storage always decreasing, the options can be numerous. Since this is a web-hosted browser application, I would personally recommend using cloud storage. Although pricier, it gives developers remote access to data from any location and very strong security from big name hosting companies. The downside of cloud storage is pricing and recurring fees. If cloud storage is not preferred, and the server is hosted in house, then I would recommend a solid state drive to offer fast data retrieval. Since hard disk drives are also fairly cheap, it would also be a good idea to use one for backup storage in the event of data loss or a systems breach.
4. **Memory Management**: Given the nature of our application using browsers clients, and the rendering of images across many game instances, it is important to have more memory than needed for consistent service. If we assume 8 mb images are being held and rendered across the targeted 4000 simultaneous instances, this assumes that we need a minimum of about 32 gigabytes of RAM to hold all instances. If we wish to prepare for the best case scenario, then I would recommend acquiring 150% of this number, since RAM is most likely the fastest and easily available resource. If rapid growth is expected and the amount of RAM is taking a toll on the budget, I would recommend using virtual memory if storage is abundant. The goal is availability of the service, so ensuring that memory leaks or anything that might crash the application are avoided is a primary focus.
5. **Distributed Systems and Networks**: Considering the desire for communication across various platforms, there will need to be a certain framework development. Most likely, client-server is the most efficient design for a web based gaming application. This implies that both an application server runs to connect with the client application and a database server runs to store user information and interface with the application server. If effort is focused on developing browser compatibility for the primary browsers which are used across the three main operating systems (Chrome, Firefox, Safari, etc.), then this will cover a majority of potential users. This means that users won’t have to directly connect with each other, and only need to query the server through the browser. If the game is intended to have it’s own mobile application, or desktop application, then a three-tier architecture may be in the cards to create a third party application to interpret requests and data transmissions to maintain connectivity.
6. **Security**: In order to keep the system protected and secure, there must be a form of username password and authentication to login. Since offering multiple platforms can open up the potential avenues of attack, it is important to be as secure as possible. It was mentioned above the distinct separation of application and database servers existing as an added level of protection against data breaches. Protecting user data is also a priority, with encryption as the default method of security. All data pertaining to users and profiles must be kept encrypted in storage to prevent leaked information from being readily understandable. This goes along with proper backend security, and also ensuring any sort of storage or server hardware is protected under lock and key. Some practices that may increase security levels are email verification during account creation, routine re-encryption or password changing for both users and administrators, and potentially even a licensed third-party security software that monitors the integrity of source code. The application could also provide text reminders to users to maintain healthy practices, such as “don’t share your password” and also “ensure your firewall is activated”.