

Draw It or Lose It Game Application

# **CS 230 Project Software Design Template**

Version 1.1

## 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 |
| --- | --- | --- | --- |
| 1.1 | 06/20/2021 | Gary Clark | Initial Software Design Doc |

## [Executive Summary](#_sbfa50wo7nsh)

With this design document, we can present the first working prototype for The Gaming Room’s web-based application. Throughout the design process we have maintained principles of secure and efficient code, while working to create a product to fulfill the desires of our client. These desires included creating functional code to maintain the uniqueness of each game attribute such as players and teams. We were also tasked with ensuring only one instance of each unique game is present within the system memory at a time. The end product resulted in a functional system to maintain the desired environment on which the game will run. This environment contains unique identifiers for each entity present with the game as well as methods to maintain the integrity of the game. The code within this proposal is easily replicated and provides easy access to build in new features. Certain aspects of this proposal remain undecided, but this will serve as a strong foundation for the staff at The Gaming Room to work upon.

## [Design Constraints](#_2et92p0)

This project presented us with multiple constraints of which we had to satisfy. These constraints represent the requirements set for by the client and this section will provide an overview of the implications of these constraints in the design process.

The first constraint presented was with the specific attributes of the game this system is designed for. We had to work in classes for each major attribute present such as unique games, teams, and players. While this can sometimes lead to a complicated design system with many different properties and attributes to track, we were able to assign each class with the same common identifiers of id and name. By allowing these classes to inherit the same elements, it will be much easier to move this system to different platforms or add and modify existing features.

The next constraint we had to contend with was within the integrity of the game. This integrity manifests within the unique identifiers that each entity must have, and this process ensures that no duplicate entities can exist. To again maintain the portability of the code, we created logic to search and check if an entity exists with certain attributes before creating a new one. The logic is the same across the classes with the only difference being the list that the system is looking through.

Finally, we were tasked with creating a system which only permits one instance of a game in the memory at any given time. In this system there exists a class of which the sole purpose is to track and update each instance of the entities. This class cannot inherit from the other classes because it would create vulnerabilities that could lead to the manipulation of data. The class can only read the data from the other classes and cannot modify it.

## [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 diagram below represents the system model which we created, including various requested attributes of certain game elements. As desired, there exists three main components to the architecture of the game system: game, team, and player. All these elements must contain certain identifiers to prevent duplication which are listed within the Entity class. This class is the parent class to the three elements which not only makes the code more efficient, but more secure as well. The elements all inherit the “id” and “name” attributes from the Entity class which allows us to track unique identifiers for all these critical elements. Based on the desired specifications, each team will consist of players and each game will consist of teams. This relationship between sibling classes is represented by the connection between them. There exists a game service class which does not inherit from the Entity class but does share a relationship with the game class. This will ensure that while many games can exist at a time, there can only be one instance of each unique game. The Program Driver class contains the methods to drive all the game classes and the singleton tester class ensures the game service class is working properly with no duplicate games.

****

## [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** | The MacOS platform supports various methods of cloud-based hosting some of which can integrate with Apple’s iCloud. They also offer a tool to create and manage servers called macOS Server which costs $19.99. They allow for Apple hosted distribution on the Mac App store, but other platforms must be managed by the developer. Many of the tools on this system are restricted to the MacOS platform making it non-ideal for hosting a large and diverse client base. | Linux is an open-source platform supporting Apache and Nginx web servers. It allows for more freedoms to developers and administrators than other proprietary systems. It is not a very common OS to be targeted by cybercriminals, but many software engineers are not experienced in this OS. As such it can be more complex to maintain and update. The OS is very lightweight on system hardware, but many professional programs do not support Linux. This OS would be ideal for eliminating licensing costs while sacrificing ease of use. | Windows is a well-established OS with long term support and an intuitive GUI. The ease of use, rapid driver updates, and automated system updates affords developers a streamlined deployment process. This access comes at a high licensing cost on a per user basis. On a security level, this OS can be prone to user errors and an integrated interface is a potential point of attack. The Windows platform is ideal for teams looking for long term support and prompt updates. In turn, this makes it an easier platform to scale up operations while coming at a greater cost. | Mobile devices are lower powered machines making this platform not suitable for hosting large scale servers. While Android is open-source, iOS is not, and server hosting capability is restricted to the former. In order to host a large client base, one would have to utilize cloud-based systems such as Google Cloud and licensing fees will vary on the size and scale of the project. |
| **Client Side** | MacOS is one of the most popular desktop operating systems among consumers. The native web browser, Safari, can interpret content based around HTML, Java, and CSS. This platform also supports other common web browsers such as Chrome and Firefox. Running a web-based application on this platform will require little modification from the development team, expediting development time. | The strength of Linux lies within its open-source nature. This provides excellent versatility for users with many popular web browsers having versions compatible with this platform. Development for this platform would require little modification, with the exception being additional threat detection from source code modifications. | Much like the other desktop OS counterparts, Windows compatibility with web-based applications is exceptional. It is a more diverse OS than either macOS or Linux which will require the development team to implement APIs for a greater number of web browsers, but this is offset by the ease of development on the Windows platform. | While mobile platforms do contain built in web browsers, the limited processing power and lower quantities of RAM make it inefficient to access an application through these applications. To support the greatest number of mobile devices, an app would have to be created that communicates directly with the web-based servers. This will add development time and cost as more developers will be required to work on the different mobile operating systems. |
| **Development Tools** | Many development tools exist for this operating system. Xcode is Apple’s native IDE which supports a myriad of programming languages. Other IDEs are also available on Mac such as Netbeans, Visual Studio, and Eclipse. Mac OS also supports MySQL which is one of the most powerful database platforms available. Licensing costs will depend on chosen tools for development, but Xcode is included with the operating system. | Linux supports many popular script languages such as Python and Ruby. Support for MySQL gives free access to powerful database tools and access to IDEs such as Atom and Eclipse gives developers a wide range of tools to create applications. There exist no licensing fees for most Linux-based software which can cut down on development costs. Development on this OS will require a team experienced in the Linux OS which may be more difficult to find than other platforms. | As with the other desktop OS’s, Windows supports development on many programming languages. Microsoft’s Visual Studio is one of the premier IDEs available and Windows allows for seamless integration. Other IDEs such as Eclipse, Atom, Netbeans, and Android Studio are also supported giving developers freedom of choice. Licensing fees vary, but full access to Visual Studio starts at $45/month for small to mid-sized organizations. The benefit to utilizing this IDE comes from the native integration with Azure Cloud Services which can lead to a faster deployment. | While development is extremely limited on mobile platforms, many tools exist to develop applications compatible with these operating systems. Android Studio allows for open-source application development on the Android operating system and the built-in emulator allows developers to test their programs using virtual mobile hardware. Development for iOS is very restricted to Xcode which is only available on the Mac operating system. To support all platforms, multiple development teams would be required adding to the cost and time required to complete the project. |

## Recommendations

1. **Operating Platform**: Microsoft Windows is the most used desktop operating system in the world with a greater than 75% market share and it will provide the greatest access now, as well as the greatest expandability later. This platform supports a variety of development languages which allows for easy porting of the application to other platforms. The versatility of this operating platform is further exemplified by Microsoft Window’s 30% market share of the server category. A web-based system will be crucial in allowing for multiplatform use and the ability to use the same operating system for development, execution, and distribution provides the greatest opportunity for success. The cost of this platform is more than open-source platforms such as Linux, but the cost will be offset by the amount of time saved on development and deployment. If we can roll out this application quicker and to more users our revenue can increase exponentially.
2. **Operating Systems Architectures**: As an x86-based architecture, Windows easily integrates with both 64-bit desktop applications and 64-bit mobile platforms. This portability is key for a fast deployment to the mobile market. The server platform offered by Windows supports Microsoft SQL and allows for seamless integration with the Microsoft Azure cloud platform. Scripting is easy to manage and install on Windows which utilizes the ASP and ASP.NET frameworks. With tools such as Visual Studio supporting many of the most common programming languages, our development team will have immediate access to a plethora of tools and support. This OS is a closed system, which allows for less customization as a trade off for increased fluidity and technical support.
3. **Storage Management**: While working in the Windows Server environment, the team must make use of the multiple available methods the manage storage. All data vital to operations will be storage on NAS servers configured to RAID 1 for redundancy. Player and user data will take up the bulk of the stored data, but we can alleviate the stress on the server side using localization techniques for some of this data. The unique identifiers for user accounts must be stored server side, but lesser data such as the users profile picture or screen name can be stored client side. With many accounts expected to be created and removed constantly, it may be beneficial to utilize a storage management provider through the Windows Storage Management API. Storage should not be too much of a concern on this platform with ample space available through the integrated Microsoft Azure.
4. **Memory Management**: The built-in memory management on Windows divides available memory into two partitions. The first partition is reserved for kernel operations and the second partition is reserved for application operations. This separation of memory usage makes fatal system errors from memory mismanagement unlikely, but this does not prevent the application crashing from memory mismanagement. Keeping this in mind, we must ensure the console clears out game states once completed to prevent a compounding of game states taking memory resources. For redundancy, we can utilize dynamic memory allocation and set parameters to ensure the application never uses 100% of system allotted memory. During high traffic hours it would be best to have multiple secondary servers on standby to handle overflow traffic from the main server. Finally, we must thoroughly stress test the application framework with particularly close attention paid to the location of any memory leaks. If we maintain the current state of the framework which is built in Java, we need not worry as much about leaks because the JVM Garbage Collector automatically managements memory allocation within the application.
5. **Distributed Systems and Networks**: To achieve the project goal of deploying Draw It or Lose It on multiple platforms, we will need to utilize various distribution techniques. The basis of our deployment process is centered around the web-based nature of the application. Desktop operating platforms will be able to access the application through any web browser with Java enabled. The expansion to other platforms including mobile systems becomes somewhat more difficult. We can begin to address this challenge by defining the nature of our framework. There is a client side and a server side. These two sides work together to enable successful runtimes of the game. The client side will have to store some localized player data, game assets, and instructions for communication with the main server. This data can be packaged within mobile applications compatible with Android and iOS using the same infrastructure as the web-based version. Porting the application to these platforms can be done by smaller development teams since our Java application is supported by Android Studio and XCode. On browser-based instances the client side is not required, and all processing will occur on dedicated servers. The communication that takes place between the client and the server will be handled via various transport network protocols such as TCP and UDP. By using TCP in conjunction with IPv4 during the transfer of user data, we can maintain the confidentiality of this data. The bulk of the client-server communication will be handled with DNS protocols, but other internalized systems should use DHCP and FTP. In browser game instances, we can utilize HTTP, but this method should have a hard separation from internal systems as it lacks encryption capabilities. On the server side, we will need multiple servers for different aspects of operation. There will be one main client facing server which handles incoming data from clients and runs most normal game operations. This main server will communicate with storage servers containing game and user data via the FTP. To ensure the portability of the application and to allow for easy update deployments, we must also have a development server which can double as a test bench for internal quality assurance evaluations. Finally, we must consider the necessity for redundancy in the event of unforeseen circumstances. As mentioned before, data will be stored in a RAID 1 configuration and there will be multiple auxiliary servers to compliment the main server. These auxiliary servers can act as the main server in the event of an outage to maximize the uptime of our application.
6. **Security**: The one major downside to development using Windows is that it is susceptible to security breaches stemming from hacking or human error. However, there are some steps we can take to reduce security risks. In a web-based game, many of our systems will be public facing by necessity which can make for an easy target for cyber criminals, and we must employ modularity to combat this risk. This is exemplified within our plan for multiple servers where the public facing nodes cannot send or modify data on internal nodes except for select whitelisted operations. In doing so, we can create a wall of sorts between critical systems and public facing systems. Throughout the network there are many instances where encryption can be deployed to further secure sensitive data. Besides using IPv4 to handle user data requests, we can encrypt the user data on the storage servers to protect it should a threat actor breach our internal systems. While Windows is notorious for its vulnerability, it still possesses many useful tools to help secure or “harden” the servers. We can follow the Windows Server Hardening Standards which includes guidelines for security auditing, firewalls, account policies, and updates. These methods can help solidify the technical aspects of security, but the human aspects are just as important. On the development server, two factor authentication must be enabled, and regular password changes must be required. The security of a network is only as good as the people using it. Keeping these practices in mind, we can maintain the security of this application and we also have vast support available from Microsoft.

**References**

*Operating System Market Share Worldwide*. StatCounter Global Stats. (n.d.). https://gs.statcounter.com/os-market-share.

*Server Hardening Standard (Windows)*. IT Security. (2018, July 2). https://security.uconn.edu/server-hardening-standard-windows/.

Carpenter, T. (2011). *Microsoft Windows server administration essentials*. Wiley.