

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 | 03/20/2022 | Jeremy Morrison | Update the executive summary, design constraints, and domain model explanation. |
| 1.0 | 03/31/2022 | Jeremy Morrison | Update Evaluation of platforms |
| 1.0 | 04/14/2022 | Jeremy Morrison | Update Recommendations |
|  |  |  |  |
|  |  |  |  |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming room would like to design a web-based version of a gaming app and is unable to develop the environment. The Gaming room would like certain requirements met such as the ability to have one or more teams play, each team will have multiple players, game and team names will be unique to allow users to check whether a name is in use when choosing a team name, and only one instance of the game can exist in memory at any given time. In order to address these requirements, we will need to manipulate existing code and address constraints.

## [Design Constraints](#_2et92p0)

* Cost to train staff on web-based system
* Cost and training for IT staff to support web-based system
* Training materials may be different on web vs. the gaming application previous system
* Optimizing for all browsers and systems

## [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 ProgramDriver class contains the main method, it is the class that runs the instruction for the software project. The ProgramDriver class calls the public static main method which obtains an instance of GameService and SingletonTester. Once GameService is called we have a private static method list games which contains a list of active games. A private static method nextgameid which holds the next game identifier. A private static gameservice instance which instantiates the gameservice variable that is accessible throughout the class and is the only instance of this variable allowed. Then we have the private gameservice constructor which keeps other methods from creating new instances of gameservice. A public static gameservice getinstance method that returns the gameservice instance to other methods. A public addgame method that will check the current game list and return current game or add game if the name is not found in the current game list. A getgame index method to return a game at a specific index. A getgame method to search the list for a game id and return current game id instance. A getgame name method to search the list for a game name and return the game instance with that name. Finally a getgamecount method to return the number of currently active games in the list. The gameservice class has a zero to many relation the game class meaning there can be zero to many game instances communicating with the gameservice method. The game class contains a private method game that hides the default constructor so other instances are not created. Then we have a public game constructor that holds an id and a name used to identify the games. A getid method to return the game id, get name method to return game name, and a tostring method to return game id and name in a string. Next is the team class that has a zero to many relation to game because there are zero to many teams allowed to communicate with the game class. The team class has a team method that holds the team id and name, getid method that returns the team id, getname method to return the team name, and tostring method to return team name and id in a string. We then have a player class that has a zero to many relation to the team class because the team can have zero to many players in it. The player class contains player method to hold player id and name, getid method and getname method to return id and name, and to string method to return these variables in a string. The Entity class is inherited by the game, team, and player class. This means that all three of these classes contain information obtained from the entity class rather than all three of these classes creating new variables for names and id’s they can all call the information contained in the entity class. The entity class contains methods for the id and names and to string. The singletontester class has a properties association to the programdriver class because the singletontester class is a test class to test the functionality of the main method.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

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. keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is a higher cost system but similar to Linux it has great stability and performance. Easy to use graphical interface.  It is not ideal to scale and doing so requires intimate knowledge of the system. Difficult to integrate into a larger environment. Less hardware and software options but less downtime and updates. Top of the line hardware with great security. Hosts run Apache servers similar to Linux allowing for basic web code to run smooth. Easier to set up bootcamp and install/run dual operating systems than windows vm. | Free and open source, this is the more cost-efficient solution.  Also, the most stable and best performance. Convenient and fast ftp tools. It is much safer than most options in terms of security. More difficult to operate requiring the use of terminal. Linux is compatible with most web hosting software and hardware but not compatible with windows applications creating issues if the company requires the use of a windows application on server. Linux does support a lot of scripts. Migration issues. | History of issues such as blue screens, system crashes and bugs. Higher cost. Easy to use graphical interface. Convenient and fast ftp tools. More hardware and software options but frequent updates and downtime. Often slow and choppy. Cheaper than mac, more spendy than Linux. Can run applications that use ASP, .NET, Microsoft Access, MSSQL databases that are not able to run on mac or Linux servers. More backwards compatible. Able to stream Xbox or PlayStation to windows. | Various applications for android mobile devices to code and run servers from devices. Low cost. Simple to do. Best done through cloud hosting but can be done from device. Connection issues, server configuration issues, hardware constraints generally CPU and RAM, security issues. Good for smaller tasks not scalable. Devices can be jailbroken allowing outside applications. Short battery life.  Mobile must be developed separately for each system. Offers a broader audience. Less cost to upgrade |
| **Client Side** | More streamlined user interface/simpler and similar on all mac devices. Single ecosystem allows for seamless assimilation between mac and iOS devices. Simple easy to use graphical interface. Less susceptible to security issues than windows. Smooth operation and specialized in multitasking capabilities. More expensive than the alternatives. Limited options regarding software and hardware customization. Less options for gaming titles, apps, software. More processing power. Cost of repairs is a con here. | Free and open source. Difficult to operate without experience. Easy to install. Works on any operating system. Works with any web browser. Many free and open-source features and software options similar to mainstream software such as Microsoft word or photoshop. Uses very little system resources and space. Cons are having to find work around software, learning curve, some software solutions are not as powerful, running server-side programs, not a gaming system. | More complicated workflow, not as easily synced with other platforms and devices. Simple easy to use graphical interface. More customization features regarding hardware, software, and applications. More game titles. Designed for gaming experience. More support needed for technical issues. Larger community of users. Backward compatibility. Easy and affordable repairs. | Smaller screen size. Very convenient to users, portability. Difficult to implement many things in a single window. Limited offline functionality. Personalization. Portability also means interruptible; applications must be designed for interruption and make it easy to recover from lost connection. |
| **Development Tools** | Major development tools and languages for mac would be: atom, delphi, cloud 9, embold, kwatee, zoho creator, eclipse, pycharm, visual studio, html, css, javascript, php ruby on rails. There are a wide range of tools and languages for mac development that range from free open source to subscriptions and software fees. | Major development tools and languages for linux are atom, delphi, cloud 9, embold, kwatee, jira, azure, eclipse, html, css, javascript, python, java, php, ruby on rails, pycharm, eclipse, visual studio, etc.. all platforms have many tools available and languages. | Similar tools as the others including azure, axure and all similar programming languages and there are many other options available for all platforms. | When developing for mobile it is a little different than developing for the other platforms because each operating system must be developed separately which can be done with different tools or developed together using cross platform development tools such as Xamarin, titanium, unity3d(good for developing mobile games), phonegap, sencha, qt, corona, react native. There are many cross platform options that allow you to develop for multiple mobile platforms simultaneously. |

## 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 a cloud server solution such as Amazon Web Services, Google Game Servers, or IBM Game server hosting. Operating server hosting in the cloud allows maximum up time, great security, data backups, flexibility, mobility, and unlimited room to scale with the consumer base without having to guess the magnitude of the consumers. As far as Operating Systems I would suggest Windows based hardware for the vast number of resources available to Windows users and their affordability. Possibly hardware to test functionality of the web application on separate platforms such as mac, Linux, and mobile devices.
2. **Operating Systems Architectures**: Cloud server architecture or Infrastructure as a Service (Iaas), depends on whether it is publicly shared, private, hybrid, or multi architecture. For the game room it would be sufficient to use public cloud architecture. The Cloud service provider provides resources that are shared across multiple consumers over the web. This architecture is cost efficient, easily scalable, and requires minimal to no maintenance. The core foundation for cloud architecture is virtualization and infrastructure. The Gaming Room will have access to all server hosting capabilities virtually as if the server was hosted in house without having to possess the hardware in house. The data is stored in data centers throughout different locations keeping data safe and The Gaming Room servers up and running whereas housed servers can be interrupted through power or network interruptions, the cloud is spread out so that if one piece of the machine stops working the others pick up the slack and keep things operating smoothly. Management of server can be done from any operating system with an internet connection so whether you are using Mac, Windows, or another option you are able to access your servers from the online platform. The components and subcomponents usually consist of a front-end platform or client such as the Windows hardware, the application or middleware that allows the client to access data and storage through a web browser.
3. **Storage Management**: Google cloud offers different storage methods, for the gaming room I suggest a file system structure as that will be the most user-friendly option. The actual hardware for storage is handled and maintained by the cloud service provider. Google as well as many others use a cloud storage interface which separates each server or application but allows each to have full access to data center resources.
4. **Memory Management**: For this web application, I recommend creating an asset manager in the code that will handle the queuing of assets and report back when assets are ready. The Asset manager will queue up downloads, start downloads, track success and failure, signal when everything is done. The code will begin by queueing up downloads of assets such as images without actually downloading. Once all the assets are queued the asset manager will begin downloading the assets. The browser will typically download in parallel up to 4 connections per host. In order to speed this up we can use a range of domains for hosting the assets. Once completed the assets will be stored in cache memory and the game will receive a signal to start and begin rendering assets. As the game is played the asset manager will handle calling the assets from the cache.
5. **Distributed Systems and Networks**: The client server model defines an application structure where the applications tasks are spread throughout a network of servers and clients to enhance security and performance. The cloud server is a client server model. The servers await communication from clients and distribute their resources. Clients usually communicate with the servers using TCP/IP protocol. The user will open a browser and enter the desired URL, the browser will request the DNS server, the DNS server will look up the address to the web server, DNS server responds with the IP address of the web server, the browser will send an https request to web servers ip, the server will send over the site files, and finally the browser will render the files displaying the web application.
6. **Security**: Securing the web application will require maintaining security during development, use injection and input validation to rule out as many user input errors as possible, encrypt user data and any data flowing between services, use exception management, and of course authentication role-management and access control. Ensure that all configurations are done properly, use and redirect all http traffic to https, use auditing and logging of end user’s actions, as well as quality assurance and testing.