

Web-Based Game Application

# **CS 230 Project Software Design Template**

Version 1.1

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/17/2024 | Dalton Johnson | Initial Creation of the Project Design |
| 1.1 | 10/04/2024 | Dalton Johnson | First Edit regarding Memory |
| 1.2 | 10/17/2024 | Dalton Johnson | Second Edit regarding OS Architectures |
|  |  |  |  |
|  |  |  |  |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

The software design must include:

* The ability to have one or more teams involved
* Each team will have multiple players assigned to it
* Game and team names must be unique
* Users must be able to check name availability when choosing a team name
* Only one instance of the game can exist in memory at any time
  + Must create unique identifiers for each instance of a game, team, or player

## Requirements (v 1.1 edit to have description be more specific)

The Client is requiring for there to be unique identifiers for each game, team, and user on the platform. Additionally, the Client requires only one game instance to exist at any single moment. While there will be unique identifiers for teams, there needs to be a functionality to have multiple teams within a single game environment.

## [Design Constraints](#_2et92p0) (v1.1 edit to add more detail

The program must not allow for multiple game instances to exist at one given time, instead having only one single game instance. Additionally, there cannot be duplicate team names since teams must be easily distinguishable from each other.

## [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.

## "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.[Domain Model](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

The GameService Class contains all of the data, methods, and variables for the game. This class then uses a constructor method to instantiate an instance of the Game Class, which may have 0 or more than 1 case.

The Game Class contains a method to construct an instance of the Team Class, which may have 0 or more than 1 case, similar to the GameService Class.

The Team Class contains a method to construct an instance of the Player Class, which may have 0 or more than 1 case.

The Game, Team, and Player Classes are all children to the Entity Class, which inherit methods and variables from the aforementioned class

## [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.

V 1.1

Based on the updated information found in the table regarding the cost, effectiveness, and requirements to host a server, I would recommend Linux to host the server. Mobile seems to be somewhat lacking in development capabilities, and would be better suited for a different application to be hosted. Windows is a viable option, with variety, however Windows has several options each with their own payments. MacOS is a good selection as well, since it has an option for free development tools and has a variety of options for developers, however the open-source nature of Linux is appealing in the sheer number of tools available for use, even if they may come at an annual fee.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Strong Security  Generally Reliable and Secure  V 1.1 Edit  There is an easy to set up Deployer that can make use of multiple Deployment models to fit any need  macOS Licensing is free | Strong Security  with frequent security updates and robust permissions system  V 1.1 Edit  Since this is open source, there is some flexibility and customization that can go into the deployment of a server to fit specific needs  Linux Licensing is free | Somewhat lacking in security, and is a primary target for malware due to its popularity  V 1.1 Edit    You can manage Windows Deployment Series through the Windows Deployment services role on a server, making it simple to manage server deployment  Windows Licensing ranges from $501 - $6,155 | Weaker Security compared to other systems, and has less flexibility with hardware and computing power  V 1.1 Edit  Mobile deployment offers an advantage to provide connectivity even with limited or no internet access, and uses peer-to-peer communication, however the limited battery life, memory, and processing power is a drawback  Mobile licensing varies based on server capabilities |
| **Client Side** | Expensive  Sleek User Interface design  Exclusive to Apple Hardware, limiting Hardware Choices  V 1.1 Edit  Must ensure that the application is not reliant on any Mac-Specific software, Needs to make use of an API, considerable time and expertise must be expended to ensure the application is compatible with multiple clients | Open Source, making software less expensive  High Customization and Flexibility  Requires more Technical Expertise  V 1.1 Edit  Needs to make use of an API, since there are a considerable number of open source tools available, the cost should be relatively low considering the savings from open source software | Extensive software compatibility  Strong gaming support with DirectX compatibility  User friendly with familiar features  V 1.1 Edit  Has ample access to software development tools, however some cost may be expended towards some tools used | Software limited to device compatible applications  Limited Customization  User friendly interface  V 1.1 Edit  Have relatively limited access to development tools, which will likely result in more expenses allocated towards programming efforts, and making use of what tools are available, rather than finding the best fit |
| **Development Tools** | Primarily uses  C language  Tools include  Xcode  Homebrew  V 1.1 Edit  There are free tools to be used, or subscription tools that provide far more capabilities  Due to the large number of the available development tools, there may need to be multiple teams working on specific tasks for the best product to be achieved in an efficient manner | Primarily uses  C language  Tools include  Yocto  Buildroot  V 1.1 Edit  Linux has free and paid tools, however it seems that paid is the best option (Red hat Licensing is common with Linux)  The Red Hat Subscription would provide a wide array of open-source tools for development teams, meaning that multiple teams may be the best way to make use of all the tools provided. | Primarily uses  C language  Tools include  Windows Assessment and Deployment Kit  (Windows ADK)  V 1.1 Edit  Has a wide variety of development tools, however a good amount of them come with a payment. This can be costly if the development of the application will need multiple tools to launch the game | Primarily uses  C or C++ languages  Tools include  Xcode  Android Studio  AppCode  Xamarin  V 1.1 Edit  There is less variety in the tools available, and they require annual payment for use, but the pricing varies depending on the type of application used. |
| **System**  **Architecture**  **V 1.2** | MAC OS is based on the Darwin Kernel, which has no GUI, and is open source | Linux architecture is layered, with the hardware as the core, followed by the Linux Kernel, the Shell, Applications, and finally the User  This allows for easy debugging when necessary, if you work from the bottom up then you will know where problems are, which is important since a lot of Linux is Open Source | Windows architecture is layered, with two main components: Kernel mode and user mode  This restrains the user input, making it more difficult for important applications and data to be altered, whether maliciously or not, by the user | Mobile architecture is also layered, having hardware as the core, followed by libraries and runtimes, then applications  The components consist of the Kernel, User interface, and the Application frameworks (the frameworks contain preexisting code to be reused for common functionalities |
| **File System Components**  **V 1.2** | There are several components:  Apple File System, which has strong encryption, space sharing, snapshots, and fast directory sizing  Mac OS extended which is acceptable for traditional hard drives | Linux can use  XFS, which is the default file system for RedHat Linux  Btrfs, which offers advanced management, reliability, and scalability features | Windows can use  NTFS which is the primary system for windows, and offers encryption, disk quotas, file permissions, and compression  FAT (file allocation table) which has a performance decrease as volume increases, no ability to set permissions | Mobile uses  Android Package (APK), which can distribute and install apps, games, and middleware  YAFFS2, which is the primary system for early android devices, and supports several flash geometries  Apple File system, which is proprietary and designed for encryption |
| **Memory and Storage**  **V 1.2** | MacOS uses a virtual memory system reliant on constant uptime  Mac Storage uses several systems  Apple File System (APFS), which features strong encryption, space sharing, and fast directory sizing  Internal/External disks are used to contain the apps and information used by the device | Linux uses a complex memory system including virtual memory, Demand paging (which forms pages of virtual memory on demand and manages disk loading)  Linux generally uses logical volumes to provide storage for the user that is not on a physical disk, and they are managed by system software known as Logical Volume Manager (LVM) | Windows uses a kernel-mode memory manager  This manager allocates memory dynamically and virtually, and supports shared memory, memory-mapped files, and copy-on-write  Windows uses NTFS for the file storage system, which has features including compression, file permissions and encryption, disk quotas, and automatic restores using log file and checkpoint information | Mobile has several different management systems depending on manufacturer  Android uses ART or Dalvik virtual machine, paging, and memory mapping  iOS uses ARC, which automatically manages memory allocations  Mobile storage is often flash memory, which is internal, non-volatile memory, which is compact and slow |
| **Distributed Systems**  **V 1.2** | Mac makes use of several systems  Distributed File System (DFS)  Xgrid  Message Passing Interface (MPI) | Linux makes use of several systems  Debian, community driven distribution  Fedora, a commercial distribution from red hat  Red Hat Enterprise Linux,  A commercial distribution from Red Hat | Windows uses the  Distributed File System (DFS) to share in a tree-structure that allows users to access the same data from multiple locations | Mobile uses systems that integrate multiple resources and processing functionality into a single system |

## 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**: Windows OS

Windows is recommended due to its extensive use as a gaming platform, and its compatibility with a wide variety of software applications. It also has decent encryption for its file, memory, and storage management systems, which would be beneficial considering the large number of users that will be using the application. While this option may not have as much flexibility and customization as other Operating Systems, overall Windows is functional with what it is needed to do, and does it exceptionally well.

1. **Operating Systems Architectures**: Microkernel Architecture

The Microkernel Architecture will allow the game service to remain reliable with the high modularity of this design, reducing the chance that the game would crash and remain offline for an extended period of time.

1. **Storage Management**: Magnetic Disk

Since the game is browser-based, there will not be much need for storage other than the application itself, since each new game will be a new instance wiping any previous data clean. Virtual memory is an option for the system as well, which will not be an issue since the game will have permanent uptime, and should have various redundancies to ensure that remains to be the case.

1. **Memory Management**: Swapping

Swapping will be a sufficient memory management technique for the application, since it allows for faster processing speeds, and will be able to prioritize the game processes for execution.

1. **Distributed Systems and Networks**: Application Programming Interface (API)

An API would allow for different systems to communicate, there is a wide variety of different APIs, which should be looked into further later on, but the most popular game API is RAWG, which should be considered.

1. **Security**: Authorization

Use password authorization for user accounts to ensure that their data is secure, as well as encryption on various levels of data storage and management. This ensures that even if one layer of the application is compromised, there are redundancies that will make it difficult to gain full control, and can mitigate the damage done in a potential attack.