

The Gaming Room

# **CS 230 Project Software Design**

Version 1.5

## 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 | 03/24/24 | Kenneth Pinkerton | In this revision initial advisements are given as well as suggestions for server/client-side implementations as well as what development tools may work best. |
| 1.5 | 04/07/24 | Kenneth Pinkerton | Refinement to how the Application can be further refined for various operating systems. |
| 2.0 | 4/21/2024 | Kenneth Pinkerton | In this revision final recommendations are addressed taking previous points and client objectives into account. |

**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)

This solution will provide a web-based application for a game that the client has developed. Specifically, the client has requested that unique teams and players be tracked by the system to know if a name has already been used in the system, however only one game instance will be permitted at any given time. In this analysis, we will review advised implementations and considerations for this project.

## Requirements

* The client has requested that there only be a single instance of a game at a time.
* The client has requested that there only be a single instance of any player or team in the system. These players may participate in multiple games but may only have one instance of their account.
* Because of the short nature of the games the system will need to be well optimized so that there is little to no lag for players to help facilitate quick rounds and good gameplay experience for the users.
* Each team will have multiple players per team.

## [Design Constraints](#_2et92p0)

* Because this will be a web-based application all of the core processing will need to be done server side and only rendered on the user’s device.
* As there will be multiple teams and players making sure that all players information is properly contained and inherited from the appropriate team will be necessary for this project.
* Because there is a heavy emphasis on team and player uniqueness principles of encapsulation will be necessary in order to achieve the clients requirements.

## [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)

* In the UML class diagram below the flow of information between various classes is described.
* The ProgramDriver is the only class that can use the SingletonTester class. The ProgramDriver is also called from the main function.
* For the different entities within the game, the Game, Team, and Player objects all inherit from the Entity object where private information is held for the entity ID and name, these can be retrieved and set by public functions within the entity class. There is also a function that allows the entity information to be converted to a string so it can be output more easily from the system.
* The Player object has a 0 to many association with the Team object, this means that a team may have no players or many different types of players. Likewise, the Team object also has a 0 to many association with the Game object.
* The Player object has unique functions attached to it so that a Player instance of entity can be created by issuing Player followed by the id and name to be associated with the correct player.
* The Team object has a private attribute that holds a list of the players associated with that team.
* The Team object also has several public functions to add additional information onto that object, namely a method to create a new team with a given id and name, a method to add an existing player object to the team by referencing the player name.
* The Game object has a private attribute that holds a list of Teams that are participating in that game instance.
* It also holds a number of public methods to add additional information onto that object, namely a method to create a new Game object with given ID and name, and a method to add a new team to the game by referencing that teams name.
* Each Game Object also has a 0 to many relationship with the GameService which maintains the game data for all of the games.
* This game service has private attributes containing a list of Game objects, methods to iterate through the GameId, PlayerId, and TeamId’s, and the GameService service itself.
* There is a private method that runs the GameService function.
* There are also public methods to add or get more information to the GameService object. These include a getInstance of GameService which will return the relevant information for each instance of GameService. AddGame which will add a new Game object to GameService by referencing the Games name. A method to getGame by referencing the chosen game’s ID. A method to getGame by referencing the selected game’s name. A method to count all of the Game objects currently associated with the GameService instance. And methods to get the nextPlayerID and nextTeamID.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac discontinued their macOS Server in 2022. Servers can still be run from a Mac or Mac Mini, but these would not be advised as running a server from a computer such as this can have issues when the server needs to be updated. In this instance the server must be taken fully offline and would not be able to be used until updates are completed. | Depending on the customers situation they may be able to rent or purchase a whole or portion of an IBM mainframe that does use Linux as it’s OS, this option is used primarily by banks and financial institutions to keep money transfers running even when the server needs to be updated/taken down. Services are offloaded onto a parallel section of the mainframe and migrated back seamlessly when the maintenance is complete. Linux has also been shown to stable, secure, and flexible in operations and can support a wide range of devices connecting to it. Additionally these also tend to be more affordable which can be a benefit to the client as these funds can then be used elsewhere.  Pricing: Linux servers $2-3K  IBM Mainframe: ~$75K  Sub-capacity licensing of IBM Mainframe: Variable, depends on utilization and can change month to month. | Windows has been another popular choice for server-side applications. However, Windows is also phasing out their Windows Server offerings and these will reach EOL in 2026.  As these are being phased out and will no longer be viable for support after 2026 it would not be advised if the company intends to maintain this application past this date. | While possible to host a server from a mobile device it is not advised as the device will not be usable for other applications due to limited resources on the hardware. It may also experience lag or other slow performance as these devices are optimized for running single applications and maintaining battery life usually. Running a server would be very energy and resource intensive for such a device. |
| **Client Side** | Many apps can be run on Apple OS’s, however specific implementations will need to be addressed as some of the core systems and syntax are different from other iterations. So it cannot be just copy and pasted to work across all formats. If only a website that the clients visit is used this would be a none issue for any iteration as virtually any device will pull up a webpage similarly as long as the styling is the same across these. Developers will need experience with working with Mac OS.  Cost: Apple products tend to have higher price points than similar Linux/Windows variants as these will often only work in Mac environments, if these are not emulated then the cost of purchasing Mac Books for development will need to be taken into account.  Time: Time constraints will likely be similar across all 3 desktop iterations for each individually, this can be shortened by having clear documentation and UML diagrams to build each off of however.  Expertise: Mac can be quite different from other iterations so having a developer that is familiar with this environment would likely be at an intermediate to advanced level if they’re familiar with other environments also. | Most Linux applications can run software that is used with either Mac OS or Windows with some possible plugins the user would need to implement. Support can be given to the user to help find suitable plugins for the Client Side application, but many Linux users may already be familiar with this extra step in the process. It is possible to include these as dependencies in the installation phase, but will need some additional work on the development side to make sure the correct dependencies are use.  Cost: Can be developed in tandem with either Apple, or Windows thanks to emulation.  Time: Similar to other environments.  Expertise: Beginner to intermediate, environment has a little bit of a learning curve for those not familiar with it, but similar features are found in Apple products, coming from Windows may be tricky as there is less emphasis on GUI. | Similar to Mac OS, Windows Apps will need to have care taken that proper syntax and patterns are followed for developing an app for the Windows App Store.  Cost: Development likely to be cheapest as most laptops could develop software as long as they are not very low end models.  Time: Similar to other environments.  Expertise: Beginner, most users are widely familiar with Windows environments and coding within these is usually not an issue. There should be very little time needed to adjust to new systems, even if a user is coming from Apple the GUI interfaces are similar, and Linux users will likely have similar experience for what GUI Linux does have. | Mobile Devices such as Android and iOS will follow similar development patterns as Windows/Linux and Mac respectively, so there will not need to be as steep as an experience curve here, however there will be a need to make sure that the app is optimized for mobile use by ensuring good battery and other resource utilization.  Costs: Since the company already has a developed mobile version for Android this will not need to be replicated, but there will be some time taken to port this over to an iOS mobile platform. The costs here may be similar to what they have already done with the Android development and may use similar technologies if they do not code directly from an Apple device, otherwise costs here would be in conjunction with acquiring Apple hardware.  Time: The time to develop this will likely be faster than developing for desktop versions as they already have a working mobile application.  Expertise: Beginner to Intermediate, as the client already has a working mobile version the skill level needed to port this over would not be as high as developing a brand new desktop version. Much of the optimization would stay the same from Android to iOS, and may just be some of the core syntax that would be different between the two. |
| **Development Tools** | Programming Language: Swift  IDE: Xcode  Impact: Since Apple has it’s own unique language that they use and a specific IDE for developing applications for use with Apple devices this can increase the costs. Xcode itself is free to Apple users, and if using on a Windows system a Mac OS X virtual machine will need to be setup specifically to develop the iOS applications. It may be beneficial to have at least 2 separate teams one for iOS and one for Linux/Windows/Android as the languages used between each of these match well together and can all use the same IDE. | Programming Language: C++, Java  IDE: VS Code  Impact: VS Code is free for developers to use and can be used to develop all other applications except for iOS applications. | Programming Language: C++, Java  IDE: VS Code | Programming Language: C++, Java  IDE: VS Code |

## 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**: With the above in mind, I would like to recommend the utilization of Mobile Applications for the client side, and Linux for Server-side applications.   
     
   Having The Gaming Room utilize an IBM mainframe for hosting their system will allow for the greatest ability to scale, maintain operability, and security for their application. This approach has the ability to work across a wide range of computing environments and can easily host the requested services with little to no issue. The main point to keep under advisement is the often-steep cost of owning a full mainframe. This can be offset by sub-leasing a portion of a mainframe from another entity, or IBM themselves. Often these costs are variable, and change based on how much data is used. This allows for further scaling from a business side as well as from a technical perspective.
2. **Operating Systems Architectures**: By choosing to have the client side of the application be for mobile devices this will allow a wide range of customers to use the app and allow more players to use the system at a time. Care will need to be taken to make sure the app is not using too much of the device’s resources so there will be some optimization concerns, but for this app that should not be a problem. Server-side Linux has been shown to be more secure, stable, affordable, and flexible than other iterations. Also, as many of the other programming languages are shutting down their support for servers the industry seems to be leaning towards moving fully towards Linux servers.  
     
   While the mainframes are optimized for security and quickness in their operations care will still need to be taken so that the system itself does not put undue stress on the mainframe from sub-optimal approaches to storage/memory management, this is even more poignant on customer devices. As these devices will often have very limited resources there will be an increased need to insure that the app will work well even if the customers network connection is slow, or their system is already congested with other applications.
3. **Storage Management**: With a Linux based system, and if the client went with subleasing a portion of a mainframe, all of these can be handled within zOS by a mainframe practitioner this will allow for the system to be scaled as needed and have additional storage allocated to the system on the fly, if the system needs to be updated the information can be cloned onto a separate partition on the mainframe to ensure 100% uptime. As these systems are used by banks and other financial institutions their security is continually updated and resiliency is built into the system with 24/7 access to the data as the default.

In zOS data is handled differently than in other OS’s. In zOS storage management is handled in series of datasets that normally are unique in structure from UNIX or other PC systems, but can be entered as a dataset within zOS as well. These datasets are files that contain one or more records, this record is the basic unit of information used by programs running zOS. To quickly access information stored within these datasets a volume table of contents (VTOC) is used that catalogs where the information is stored, this allows the data to be moved around within the mainframe, but because a record is still maintained this information can be easily accessed by simply looking at the VTOC. This acts similarly to library card catalogs organized books and people only needed to go and look up the books information in the catalog.

1. **Memory Management**: In zOS applications each user is given a block of Virtual Storage, this storage is able to expand or contract as needed by the system to maintain resources. There is not physical data storage directly tied to the specific data as all of this is done virtually through the mainframes architecture. The data itself is broken down into data sets to help manage the data that are then handled by a catalog of VTOC that is on each of the direct access storage devices (DASDs) on the mainframe. These are the physical location where data is stored, but if something happens to one there are redundancies built in where the data is stored on multiple DASDs.   
     
   When a program is first called in zOS a portion of the file at the very beginning of the transmission states how the memory will be allocated/deallocated when the process is completed. If for some reason this file is not provided zOS will request information from the caller about how they would like to handle the allocation/deallocation of memory. All of this is handled by the zOS XML parser. This is handled in a required area within the mainframe known as the Parse Instance Memory Area (PIMA), this is the working area for the zOS XML parser where it suballocates call stacks, control blocks, the tables, and trees that are used to hold the information that is being parsed. Once the given conditions are met within the specified files that are fed to the parser it will allocate and deallocate the memory needed automatically according to the instructions it has been given, as such a talented zOS practitioner would be needed to accurately tell the system how to proceed. This training however can be completed for free by going through IBM’s zOS practitioner certification.
2. **Distributed Systems and Networks**: Since zOS is designed to work 24/7/365 there would be minimal downtime because of outages with the redundancies that have been mentioned. These systems would serve as the point that all of the apps data is stored and devices would need only to connect to request the data. If mobile devices are the primary target for utilization then these devices would first need to use their built in radio antenna to connect with either WiFi, cellular, or satellite connection to allow access to the server at a given virtual address where the information may be accessed to insure security this will need to have the user log in to their account and once this information has been verified they will be able to see their games how their team is doing, and other relevant information pertaining to their account an game.  
     
   With the proposed system it would be required to have a stable internet connection to maintain proper services. As the game itself could not fully be run on the clients device like some other apps that can be run offline there would need to be warnings put in place to alert the user if the connection is interrupted, this may be able to be done by implementing simple checks within the app to state whether the internet on the users device is down, or whether there is a connection issue with the service itself, and provide steps to re-establish a connection or how to reach out for further technical assistance.
3. **Security**: With zOS the security on the physical mainframe will be similar to and may even be in parallel with other mainframes that are used by banking institutions and are typically housed in data centers around the country. Since the app may be on a mainframe that is being leased to other applications there are features built into zOS that prevents other programs from intentionally or accidentally changing each other. There are also Resource Access Control Facility (RACF) systems built in that allow practitioners to manage access to the data and can help users to reset their passwords if need be and to manipulate the data, but this access is guarded credentials must be validated before directly accessing the back end of the system. From the client side, utilizing services like multi-factor authentication (MFA) is common for many mobile applications in todays society and may even help encourage users to trust the system more from seeing this extra layer of security.  
     
   With client side it would be advisable to require the user to log in with their e-mail address, and password, biometrics that are stored on verified devices may be used that have this feature, finally as an added security measure the customer may need to enter a code that is either generated on a service similar to Google authenticator, or that is texted/emailed to the user.