

The Gaming Room

# **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 | 05/17/22 | Nathan Mailhiot | Can create a single game instance with multiple teams and multiple players. |
| 1.0 | 6/1/22 | Nathan Mailhiot | Evaluation |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.0 | 6/15/22 | Nathan Mailhiot | Recommendations |

**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)

The Gaming Room is looking to develop a web-based game application that will be able to serve multiple platforms. Currently the game is only available on Android. The Gaming Room is looking to break this game application out into a multiplayer, multiteam game that will allow users from different platforms to play with one another. The Gaming Room is seeking help with development, management, and deployment.

## [Design Constraints](#_2et92p0)

* Server Load

1. There will be an increase in users with the game now being multiplatform.
2. The server load could be broken up by region. Which would allow for less latency while playing.

* Client Development

1. Desktop development will need multiple coding languages to implement a viable product for all the platforms.
2. For MacOS the client should write the game in Swift. This would allow for the application to be native to Mac desktops and be mobile ready for iPhone.
3. For Windows platforms, the game should be written in C#. Since C# has a great libraries available, the language is known for its efficiency, and that it runs incredibly well on windows all make this the correct choice for the Windows version.
4. For Linux based machines Java would be the best language to implement.
5. If this is to be a strictly browser-based game, then using React.JS would be a fantastic way to create a GUI for this game. It also allows for the game to be multiplatform if the users web browser is up to date.
6. Mobile development would not be too difficult for this application. As it is already on Android, the IOS version would have to be written in Swift, and then the two platforms would just have to code share.

## [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 Entity class is being used as a super class. The Team, Game, and Player class are all inheriting from the Entity class. This encapsulates the Entity class and does not allow for it to be changed directly. The Player, Game, and Team classes all use the Entity class as a constructor which allows each of these individual classes to create an id and name property. They also inherit the getName(), and getId() methods from the Entity class. This allows the base properties of the class to be broken into the Entity class to not crowd all our other classes with repeat information. The GameServices class creates a one-to-many relationship with the Game class, the Game class creates a one-to-many relationship with the Team class, and the Team class creates a one-to-many relationship with the Player class. This is to ensure that we have an instance of each class created and that we followed the singleton pattern.

**"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)

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 must 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** | There are a lot of weaknesses for hosting on Mac. Mac hosting costs a lot, it requires specific hardware, and requires specialists to handle problems.  **Evaluation**  As of April 21, 2022, Apple has discontinued macOS Server. This option is no longer available. | Linux is an excellent choice as it is the most widely used. It has hardware readily available, much cheaper than other options, and proven to have stable servers. One downside is that there is a learning curve to using Linux.  **Evaluation**  An advanced physical server would cost about $1500 a year with an Ubuntu server. With an essential server only costing $225 per year. It will also host the website with built-in security features. | Windows like Linux is also an excellent choice. One downside is that it could be costly. Windows is not the most expensive option, but if the right option is not chosen it could be.  **Evaluation**  Hosting is available through Microsoft's azure. There are multiple tiers to choose from with some being as low as $7.59 a month to some being as much as $2,312.64. Windows has a ton of variations to the server of choice and has great built-in security. | The biggest advantage would be no server cost at all if running on mobile. The most difficult part is there is extreme variation in the mobile market. This would make implementation extremely difficult.  **Evaluation**  Hosting on mobile devices should not be worth the number of different hosting options you would have to go through. |
| **Client Side** | Mac devices (iPad, iPhone, MacBook, iMac, etc.) There would need to be a whole team to make sure the Mac platform would be well maintained. The staff would need to be experts in MacOS. Java, XCode, Swift.  **Evaluation**  One language that can be used to ensure that the application is compatible across all web browser platforms is to write it in JavaScript. | The team would need to be well versed in Linux. Need to have the most up-to-date equipment and drivers to make sure they can achieve their goals. Java  **Evaluation**  One language that can be used to ensure that the application is compatible across all web browser platforms is to write it in JavaScript. | The team would need to be experts with windows machines. Make sure all the software is up to date. C#, C++, Java.  **Evaluation**  One language that can be used to ensure that the application is compatible across all web browser platforms is to write it in JavaScript. | The teams would need to be off shoots of the windows team and the Mac team. They would need to be able to be up to date on all software and security. Swift, React Native, Java  **Evaluation**  The mobile versions need to be written in Swift, React Native, and Java |
| **Development Tools** | Visual Studio Code, Eclipse, IntelliJ  **Evaluation**  Eclipse and visual studio code are both free services. IntelliJ is priced at $499.00 a year. XCode is another source that can be used, and it is $99 a year. | Visual Studio Code, Eclipse, IntelliJ  **Evaluation**  Eclipse and visual studio code are both free services. IntelliJ is priced at $499.00 a year. | Visual Studio Code, Eclipse, IntelliJ  **Evaluation**  Eclipse and visual studio code are both free services. IntelliJ is priced at $499.00 a year. | IOS, XCode, there is no good option for mobile.  **Evaluation**  XCode is priced at $99 a year. |

## 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**: For deploying the application I would choose a Linux based server. It allows for scalability, performance, security, and affordability. The only difficulty that would be present in this selection would be the learning curve for the staff. The benefits outweigh the learning curve though, and once it is learned it will be able to easily be maintained. It will also allow the Gaming Room to expand from the current game with much more ease over time.
2. **Operating Systems Architectures**: Ubuntu has been a tried-and-true Linux based server that has shown it’s stability. It has a robust set of features, is very secure, and is the leading Linux based server.
3. **Storage Management**: Using a cloud-based platform like amazons S3 storage would be a smart choice for this game. The data can be backed up and scaled whenever it is needed. This also allows the Gaming Room to scale down when they wish as well. This allows the storage to be taken care of by AWS and for the Gaming Room to focus on other areas of their game.
4. **Memory Management**: Using Linux based memory management would allow for both physical and virtual memory. Setting the memory up in a linked allocation using the FAT method would be best for this game. This allows the memory to be allocated properly. The only drawback is that it can be a lengthy process to set up, but in the end, it allows the memory to be more efficiently allocated.
5. **Distributed Systems and Networks**: To fight server latency, server outages, and data loss. I would recommend that The Gaming Room have servers spread out across different regions. This would allow them to have better multiplayer, have backup servers, and allow for migrations. This would create a redundancy for the network, and having a redundancy allows for something unforeseen to happen and the overall health of the game not to suffer. This would be another benefit to using AWS servers as they can be set to distinct locations with low overhead.
6. **Security**: Having distinct roles for this game is imperative. The players having a distinct role that would not allow them to access different routes that should only be available to admins is absolutely 100% needed. Using something like Oauth by google would allow the player login to be handled. All that would be needed from there would be setting up the distinct routes. With the distinct routes the application should be protected. Also, with the AWS setup the data will be encrypted from end to end. Along with all these practices the information taken in from the users should also be reviewed by a validation function to prevent SQL injections.