

The Gaming Room app

# **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 |
| --- | --- | --- | --- |
| 1.0 | 09/17/2022 | Alex Wells | Initial overview |
| 1.1 | 10/1/2022 | Alex Wells | Platform evaluation |
| 1.2 | 10/15/2022 | Alex Wells | Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room needs an app that will be able to handle multiple simultaneous games, each game with multiple teams and each time with multiple players. These can be handled by created a separate game service to handle the data for the games and have each game, team, or player inherit from a base class.

## [Design Constraints](#_2et92p0)

* Only one instance of each game, team, and player can be active at once.
* Must be able to add teams and players.
* Each game can have one or more teams
* Each team can have one or more players

## [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)

All the classes for the game inherit from the base Entity class. This allows us to expand and create new classes if necessary more easily. We also use the singleton design pattern in the GameService class to ensure that there is only one instance at any given time.

**"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** | Very flexible terminal commands to make it easy to use as a server, but the most expensive option. | Same flexibility of Mac but significantly cheaper. | High availability of software to make up for less flexible terminal. | A server on a mobile device would suffer from underpowered hardware. However the mobile aspect of it could be used to power local events maybe. |
| **Client Side** | A median amount of expertise is required. Usually, the most expensive option but very competitive with windows in that category. Time is dependent on the specific application more than the platform in this case. It’s a good idea to keep in mind whether to distribute through the app store or through other means. | A high amount of expertise is required for Linux development. It is usually the cheapest option. Linux allows us good flexibility in development but may be hard to port to other systems in the future. | Windows is in the middle on cost although it is competitive with mac at times on cost. Windows requires the least expertise of the main desktop operating systems. Windows allows us great flexibility and lots of software to take advantage of. | Macs are most versatile for developing mobile apps as that allows the developer to use the same platform to develop apps for both Android and iOS. A median amount of expertise is needed for this, and it costs a fair bit to make two separate apps. Care must be taken that apps developed are able to be published on the App Store and the Google Play Store. |
| **Development Tools** | Mac can use many IDEs such as Eclipse, Visual Studio, and PyCharm, as well as many languages such as Java, Python, C/C++, and Swift as well as having access to XCode for mobile development. | Linux tools include Python, PyCharm, Java, Eclipse, Visual Studio, C+++ and many more. | Linux tools include Python, C++, Java, JavaScript, PyCharm, Eclipse, Visual Studio, Visual Studio Code, and many more. | Most IOS apps are written in Swift likely using XCode while most Android apps are written in Java likely Eclipse or IntelliJ. More recently React Native has allowed developers to develop one app for both OS. |

## Recommendations

1. **Operating Platform**: Based of the client’s specifications I would recommend the server app use a Linux based operating system. This is a cheaper option and is more robust than the others allowing greater control and ease of use as the client expands.
2. **Operating Systems Architectures**: Linux uses a base kernel to handle all commands and provides multiple layers on top for users to interact with. This granular control makes a great candidate for our needs.
3. **Storage Management**: I would recommend that the client use cloud based storage management. This is a more expensive option but is easy to scale with consumer demand and the client’s needs.
4. **Memory Management**: Linux uses a combination of main memory and virtual memory and various techniques to increase memory size and speed allowing for fasting programs, which will be important for our client.
5. **Distributed Systems and Networks**: By using the client server pattern we are able to ensure that the server is able to communicate with clients on various systems. We do need to keep in mind bandwidth limitations and ensure that our cloud storage provider is up to the task.
6. **Security**: The best way to protect sensitive user data is encryption. This ensure that even if there is a breach the data won’t be compromised, however we must also take steps to prevent breaches. Linux is well suited to the task since software is installed in packages and rarely has kernel access, however a robust anti-virus is still a good idea.