

Gaming Room Game Service

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 1/25/2024 | Reed Murray | Added 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)

This document is meant to facilitate the development of a web-based version of ‘The Gaming Room’ gaming app. A web-based application requires a web server as a host. The scale of server hosting will depend on the expected active daily users. A Java-based application will meet the networking needs as well as the requirements below.

## Requirements

* A game will have 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.
* Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

* Anticipating using Java programming language. Java is object oriented, which parallels the structure of a game making it well-suited to game application development. Java also has extensive network-related libraries, which gives us several options to choose from when it comes to communication to the webserver.
* Implementing the Singleton design pattern will accomplish unique identification of game instances and allow only one instance of the game to exist in memory at a time.
* Implementing the iterator design pattern to check game and team names will ensure that a name cannot be held by two different entities at any given time.

## [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 GameService class makes use of the team, player, game classes to create and run instances of the game, while tracking teams of players. The game, team, and player class all inherit from the Entity class, because they can be all considered an Entity in the event of a game being played. Encapsulation is used to ensure no data members are changed unnecessarily. For example, private constructors ensure that an empty entity cannot be made.

The program driver creates an instance of GameService and tests the code to ensure unique game identifiers.

**"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 has to 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** | A Mac is generally a more expensive machine, making it not well suited for use as a server. A drawing game can be easily hosted on a less-expensive and less-powerful machine. | Linux has a highly user-controlled system, meaning maintenance of the server would be easy and cost-effective. Linux is open source, which makes it free and means it has a lot of documentation and public trouble-shooting. | Windows is a more expensive option to host the server. It will need to run on a PC and could be considered “overkill” for a web-based application. Windows is not open-source. | An android mobile device can be used as a web-server host with the right tools, however it is limited in ability and could be out-grown if the website has enough traffic. |
| **Client Side** | Since the game is web-based, it will be accessible from any operating system that can run an internet browser. | Linux is a commonly supported OS for gaming. Developing games for this OS is going to be similar cost to windows. | Windows is a commonly used operating system meaning many programmers have experience with it. Windows is not open source, so it must be purchased for each machine it needs to run on. | Mobile development could be accomplished with two paths: Allowing users to access a mobile version of the website (more cost effective) or developing a mobile app to go along with the website (less cost affective, more popular consumer choice). |
| **Development Tools** | Java can be written in many IDEs, and is not machine dependent, meaning it can be compiled on a MAC. | Linux allows full control of the file system, making development easy to control as well.  Linux is a less intuitive system that requires more control on the user’s part. This means a well experienced Linux user would lower the overall time needed to develop. | Windows is very versatile and will run all the commonly used Java IDEs. Windows is consistent and intuitive to use. | Building software on a mobile device is not recommended. Desktops/laptops have more processing power and memory, and are not more expensive. |

## 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 recommend a windows system to develop and host the application. The resource cost is low for a drawing game, meaning it can likely be run from one or few machines. The webserver can be hosted on the same machine, or to make it more modular a raspberry pi can be used to host the website. Windows is easy to use, and almost everyone has experience with it.
2. **Operating Systems Architectures**: Windows can be used on a personal computer with a processor, hard drive or solid-state drive, RAM, motherboard, and power supply. I recommend a solid-state drive because the cost is minimal for noticeable efficiency in reading and writing to non-volatile memory.
3. **Storage Management**: Solid state drive for storage of application hosting files, and constants in the game. Scores and personal player data can be stored using a cloud based relational database such as Azure. This facilitates a generally reliable and hardware free storage management system. Cloud based storage is cost-effective, because you only pay for what you use.
4. **Memory Management**: An important aspect of porting the game is that memory addresses are being freed upon completion of a round or game. The memory being used to draw the pictures should be freed immediately when the drawing is gone. Leaving objects in the game-space when they can’t be seen is an easy way to cause a memory-leak. Java uses automatic garbage collection for memory management, making it easy to accomplish this.
5. **Distributed Systems and Networks**: I recommend using Java to develop the distributed application, due to the frameworks and libraries which can be used to distribute to the web. When distributing to a website, the hardware involved is the client side, the game server, and the web server. Using a RESTful API incorporates the necessary components to allow these sides to communicate. Using a secondary application which checks the IP/port, outages can also be tracked and detected to minimize the downtime.
6. **Security**: Using Java incorporates robust language-based protection. Authentication can be implemented to make sure users are using the application as intended. Windows is not a secure-by-default system, so it should be well inspected for security concerns often. I recommend using a higher-end processor than necessary for the server, because new CPUs by AMD and Intel have protection against buffer overflow attacks. This kind of attack is likely the easiest way to crack the game application. The processor prevents it by not allowing the return address to be executed as code.