

Module 2 Game Service

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.gjdgxs)1

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.3znysh7)2

[**Executive Summary**](#_heading=h.2et92p0)3

[**Design Constraints**](#_heading=h.tyjcwt)3

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

[**Evaluation**](#_heading=h.2s8eyo1)5

[**Recommendations**](#_heading=h.3rdcrjn)6

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/20/22 | Cody Gregory | Added team and player functionality. |

**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](#_heading=h.2et92p0)

The Gaming Room is seeking a software solution to help facilitate their web-based gaming app. Specifically they are in need of a game service that can handle the control and management of individual games, teams, and players. To best accomplish this task, we need to develop a logical foundation that meets their requirements.

## [Design Constraints](#_heading=h.tyjcwt)

* Front end programming knowledge that will be necessary to interface with the players to include:
  + JavaScript
  + HTML
  + CSS
* Web hosting as a business constraint, will cost money to keep the web app live.
* Must have only one instance of the game at any given time. Will need to employ a singleton pattern to accomplish this task.

## [System Architecture View](#_heading=h.3dy6vkm)

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](#_heading=h.1t3h5sf)

As observed in the UML diagram on page 4, the proposed solution is to start with the Entity class, from which the Game, Team, and Player classes will inherit as to reduce redundancy in code.

We can also see, left to right, that from GameService to Game, Game to Team, and Team to Player these classes have a multiplicity of ‘zero to many’ which can be thought about simply as so: a GameService can have 0 or many Games, a Game can have 0 or many Teams, and a Team can have 0 or many Players.

Lastly we can see the ProgramDriver class from which the rest of the program is called on, including our SingletonTester to verify that there is only ever one instance of a GameService as requested by the client.

**"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](#_heading=h.2s8eyo1)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Apache based server which provides fast and reliable web hosting. Intuitive interface for ease of use even if not tech-savvy. Scaling can however be difficult making expansion for more users challenging. As a proprietary big brand company, pricing for licensing is on the higher side. | Open-source and user friendly, allows users to share between multiple servers allowing for good scalability. High security. However Linux hosting is not compatible with Windows applications which can be a big downside if the client relies on windows infrastructure. As open-source is the cheapest out of the alternatives | As a Microsoft product supports all Windows based applications which are numerous. Familiar UI as most people have or actively use Windows. Less stable and more vulnerable to security threats than the alternatives. Like Mac Server as a proprietary brand is on the pricier side. | Device detection provides an enhanced user experience, automatically providing the best website view depending on the user device. Built in adaptation to provide the best display of content per individual device. |
| **Client Side** | Full compatibility with Apple products. Following in suit with the OS, Mac Server provides an intuitive UI and innate Java support for running our web application. | Open-sourced, Linux is the cheapest option of the alternatives as it is not held behind a proprietary licensing fee. Requires extensive knowledge of Unix based systems for optimal performance. | Full compatibility with Microsoft products which are extensive. Can run a Java program as simply as installing the Java Runtime Environment onto the Server just like you would for use in an IDE such as Eclipse. | Requires knowledge of Jetty as most mobile server software is based on it. Innately tied closely to Java, providing little to no changes to run our web application. |
| **Development Tools** | Supports Unix CGI 1.1 Standard, Perl, Ruby, and Python. | PHP, Python, Ruby and other Unix languages. | ASP scripts, .NET framework and other Microsoft proprietary technologies. | Based on jetty, open sourced. serves Java-based web content and is available as a set of JARs. |

## 

## 

## 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 Server OS. Windows Server provides a familiar user-interface and comes with the support of Microsoft and an extensive list of services that make it well worth the licensing fees for a company like The Gaming Room that doesn’t have an abundance of server knowledge.
2. **Operating Systems Architectures**: The Windows OS hosts the .NET Framework, which acts as an intermediary that abstracts the code and utilizes a Common Language Runtime in tandem with just-in-time compilation to convert code that is usable on the native CPU. Windows Server also supports the Client-Server architecture which is the ideal setup for a game environment such as Draw it or Lose, wherein the players are the clients, and as they play the game the make the necessary requests from the server, where things such as the bank of high definition images are stored for the game.
3. **Storage Management**: The best solution for storage in any sense lies in the cloud. Azure is Microsoft’s premiere cloud computing service that provides an immense amount of applications and services. Azure Files provides support for Server Message Block (SMB) and Network File System (NFS) protocols that can be mounted onto The Gaming Room’s on-premise server in place of hardware, and even provides support for macOS and Linux should the client desire change down the road.
4. **Memory Management**: Windows employs virtual memory to keep a multitude of processes in memory simultaneously for the sake of multiprogramming. 32-bit Windows enables addressing of up to 4 gigabytes of memory, and 64-bit Windows has a virtual address space of a whopping 8 terabytes. This affords the user a vastly increased field of play in comparison to physical memory, which by today’s standards ranges anywhere from 4-32 gigabytes on average.
5. **Distributed Systems and Networks**: The best solution for Windows Server would be to program Draw it or Lose it in a .NET compatible language such as C# or C++. This provides the application with the compatibility to run on all of the necessary platforms. In terms of a distributed system, by utilizing the Client-Server architecture failure detection is a necessity. One such method of detecting such an issue is handshaking, wherein at fixed intervals the various clients and the server send each other status signals that if not received indicate an issue that needs to be resolved.

1. **Security**: The Windows OS provides various built-in security features such as Windows Defender, there are also a multitude of security principles that Windows offers services to help secure your data and network. Transport Layer Security (TLS) and its predecessor Secure Sockets Layer (SSL) are industry standard protocols for establishing secure links between networked computers that ensure protection for both the client and server via asymmetric encryption, wherein the client and server exchange data and create a temporary encryption providing security for both sides.