

The Gaming Room

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 1.5 | 5/19/2023 | Andrew Biggs | Added code to implement the game services. |

**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)

We are developing a cross-platform game server that keeps track of unique game, team, and player names. We compare the attributes of several operating systems in how they perform as client-side and server-side, as well as each operating system’s unique toolkit.

## Requirements

The Game runs on different operating platforms.

Players running the Game on differing platforms can play together.

Each player, team, and game will have a unique identifier.

## [Design Constraints](#_2et92p0)

Each client application must display the GUI in a conformant way.

The communication between clients and server must be consistent in speed and precision across client operating platforms.

The communications must be simple enough that conversion of client data to a common language for integration across multiple platforms is fast and efficient.

## [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)

ProgramDriver uses SingletonTester to ensure only one instance of GameService runs on the server at a time. The Entity class is the parent of the Game, Team, and Player classes, as all these classes have “name” and “id” attributes along with methods to access the raw data of these attributes as well as output the information for viewing.

The Game class has a none-to-many relationship with GameService, the Team class has a none-to-many relationship with Game, and the Player class has a none-to-many relationship with Team. This is because Game Objects are instantiated only by GameService, Team Objects are instantiated only by Game Objects, and Player objects are instantiated only by Team Objects.

GameService keeps track of all Game, Team, and Player ids, ensuring that each instantiation of these Objects has a unique ID, while Game ensures Team names are unique, and Team ensures Player names are unique.

**"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** | Uses Apache.  Encrypted data transport.  Support for UNIX CGI 1.1 Standard.  Support for virtual hosting. If not using cloud hosting, Mac hardware costs more money than the other options. | A wide range of web servers to choose from.  Support depends on the server chosen. Linux web hosting is the cheapest of the options but comes with less support from the provider. | Security updating is simplified. Secured-core server for multi-layer security.  Windows Server is scalable due to its offering a wide range of back-end support and price points. | Resource allocation is distributed. Data security is a concern.  Distributed Serving with Mobile Servers (DSMS) is a relatively new technology and so inherently involves more risk and less support. |
| **Client Side** | Swift programming language expertise.  Less time consuming because testing occurs across more proprietary hardware running MacOS software. MacOS hardware costs more money.  A smaller range of client hardware. | C++ allows the client to run similar programs across differing frameworks.  Linux software is relatively versatile across hardware, and also relatively cheap as an operating system. | Universal Windows Platform (UWP) allows development across windows platforms. Potentially more time in adding compatibility with the UWP.  A wide range of client hardware. | Expertise, cost, and time will all be the highest when developing for mobile devices. Wide range of systems all with their own compatibility issues.  Includes devices running any given OS on any given hardware. |
| **Development Tools** | Scripting using Perl, Ruby, and Python.  Database integration.  Java Virtual Machine / Eclipse  XCode IDE  Swift language | C and Java are the most common  Java Virtual Machine / Eclipse  Node.js  Go  Python, Perl, PHP, Ruby | C++ and DirectX 12 are development are essential for UWP development.  Visual Studio IDE  Java Virtual Machine / Eclipse | Development tools must support Mobile operating systems: most current versions of Android, iOS, Windows Phone. |

## 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**: Linux is best for the server, since it requires no proprietary hardware nor software, is best for deployment across all platforms.
2. **Operating Systems Architectures**: Linux is controlled by the C language, which describes instructions that are further broken down into assembly to control the CPU, kernel, user applications, main memory usage, and various I/O devices. The Linux kernel is open source, which means two things for development: It will be possibly to develop tightly integrated and efficient programs, and it will also take some extra attention in development for security.
3. **Storage Management**: Using physical disks along with cloud storage will ensure greatest ease of access and protection of data. Amazon Web services provides a great range of services for storage management.
4. **Memory Management**: The Game will make calls to application instructions and data needed for program execution. Pages will be loaded into RAM using an Enhanced Second-Chance Page-Replacement Algorithm. The images for the game will be downloaded from the server and stored in a client-side cache until an image is needed. When an image is needed, it is read from the cache along with application instructions for how the image will be structured and how the game proceeds.
5. **Distributed Systems and Networks**: The server will have an application that decodes incoming data from the clients into a common format that will be understood by the server to update the current state of data in the game. The server will format outgoing data according to the operating platform that is receiving it. Additionally, the client devices need not communicate with one another. This ensures that even if the data on one client device somehow becomes corrupted, the rest of the system can continue functioning.
6. **Security**: User information will be protected on-device by using hash tables to encrypt sensitive information like passwords. The Linux server has its own security architecture that will generate and check security sequences to verify connected clients periodically. This security system will be obscured from threats by running it on a separate server with no network connectivity except to the Game server, with all data transfers closely monitored. This will protect from mimicking the security codes to mock a device and gain access to a server-client connection.

Every communication to and from the client system must be authenticated and every update to the game state that is made by the server must be authorized. This ensures every client will have a uniform game experience, and any differences will be detected

as soon as possible.

References

Atienza, ByJanus, et al. “Top 5 Programming Languages for Developing Linux Desktop Applications.” *Unixmen*, 15 May 2023, www.unixmen.com/top-5-programming-languages-for-developing-linux-desktop-applications/.

Enock. “The 10 Best Open Source Web Servers for Linux.” *FOSS Linux*, 29 Apr. 2021, www.fosslinux.com/46032/best-open-source-web-servers-linux.htm.

Ghodselahi, Abdolhamid, et al. “Concurrent Distributed Serving with Mobile Servers.” *arXiv.Org*, 16 Nov. 2019, arxiv.org/abs/1902.07354.

Stevewhims. “Windows Game Development Guide - UWP Applications.” *UWP Applications | Microsoft Learn*, learn.microsoft.com/en-us/windows/uwp/gaming/e2e. Accessed 19 May 2023.

*Web Hosting - Images.Apple.Com*, images.apple.com/server/docs/Web\_Hosting\_TB\_v10.4.pdf. Accessed 19 May 2023.

“Windows Server 2022.” *Microsoft*, www.microsoft.com/en-us/windows-server. Accessed 19 May 2023.