

Draw it or Lose it

# **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 | 09/08/23 | Darren Nason | Creating the source code in accordance to the UML diagram |
| 1,1 | 09/25/23 | Darren Nason | Adding evaluations, recommendations based upon Client/Server-side development and Development tools |

**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)

Gaming Room has encountered difficulties setting up the development environment for this project. Through proper programming practices with the use of singleton design pattern and unique identifiers for the team and players lists we will deliver a fully functional application.

## Requirements

Technical Requirements are:

* 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 to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* Web-based application – this means that the program will be platform independent and have the ability to be accessed by a wide range of devices.
* Single instance in memory
* Name validation and feedback – this allows the user to choose a team name and we need to come up with a friendly message to inform the user if the team name is already in use

## [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 the parent class through the OOP of inheritance passes down to the child classes of game team and player. Also identifying the relationships between all these classes are as follows there can be many players to a team, many teams to a game, and last many games to the game service class. Abstraction comes into play as we move from the general entity class to its specific child classes. Each child class further refines and specializes the properties and methods inherited from the entity class. This abstraction allows us to model the core characteristics of games, teams, and players while maintaining a clear separation between common attributes and those unique to each class.

**"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** | **Characteristics:** MacOS, Xcode, and Intel or M1 processors which provide decent processing power, storage, and memory  **Advantages:** Ease of use, Unix-Based System, compatibility, and security.  **Weaknesses:** Limited hardware options, cost, and scalability | **Characteristics:** distribution variety, open source, command-line interface, and stability and reliability  **Advantages:** cost effective, security, scalability, and software ecosystem  **Weaknesses:** Learning curve, compatibility, fragmentation and support | **Characteristics:**  Windows Server Editions, GUI, Active Directory, .NET Framework  **Advantages:**  Ease of use, .Net Support, and Microsoft Ecosystem  **Weaknesses:**  Cost, Resource Intensive, Updates and Reboots, and limited open source | **Characteristics:**  Limited Hardware Resources, Battery-Powered, platform variety  **Advantages:** Portability, Local Data Processing, Edge Computing  **Weaknesses: Limited resources, reliability, network dependency, scaling issues, and battery life** |
| **Client Side** | **Cost:** Development tools and licenses, testing devices  **Time:** Developmenttime, compatibility testing  **Expertise:** Mac dev knowledge, UI/UX design, accessibility, version compatibility | **Cost:** Development tools and licenses, testing devices  **Time:**  Compatibility Testing, Dependency Management  **Expertise:** Linux dev knowledge, compatibility, command-line interface, and security | **Cost:** Dev tools and licenses, testing devices  **Time:** compatibility testing, UI adaptation  **Expertise:** Windows dev knowledge, security, UX, and performance optimization | **Cost:** Dev tools and licenses, and device acquisition  **Time:** Compatibility Testing, and UI adaptation  **Expertise:** Platform-specific development, UX, performance optimization, and security |
| **Development Tools** | **Programming Languages:** Swift and Objective-C,  **IDEs:** Xcode, Eclipse, IntelliJ, VS Code  **Additional Dev tools:**  Interface builder, Core data, and TestFlight, Git, Docker | **Programming Languages:** C, C++, Python, Java, and Rust  **IDEs:** VS Code, Eclipse, IntelliJ, Vim  **Additional Dev tools:** Node.js, Git, Docker, GDB debugger | **Programming Languages:** C#, C++, Java, Python, JS/TS  **IDEs:** VS Code, Visual Studio, Eclipse, IntelliJ  **Additional Dev tools:** ASP.NET, SQL Server, MySQL, Git, Docker | **Programming Languages:** Swift, Objective-C, Java, Kotlin, React Native, Flutter  **IDEs:** Xcode, Android Studio  **Additional Dev tools:** SDKs, Git, App Store Connect, and Google Play Console |

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
2. **Operating Systems Architectures**: Flexibility within Windows Server to tailor to different use cases, Also if you plan on managing the app windows has a user friendly graphical interface to do so.
3. **Storage Management**: MySQL or PostgreSQL
4. **Memory Management**: Java Garbage collection-this is an automatic memory manager within Java that identifies and reclaims memory that is not in use.
5. **Distributed Systems and Networks**: We can accomplish this by implementing client-server architecture where one central server will act as the hub for managing sessions, data, and game logic. We could use data backups to handle server outages. Also, we can handle network outages by implementing a local data cache on client devices.
6. **Security**: In order to make the application secure we can apply industry standard protocols such as HTTP/HTTPS for web communication. To authenticate users that come into the site and play the game we can use JWT or OAuth 2.0. Through utilizing input validation, we can ensure all user inputs are valid and sanitized to prevent cross-site scripting and any other injection attack. To secure the data used within the app we can use encryption mechanisms within the database we decide on like MySQL’s Transparent Data Encryption or PostgreSQL’s pgcrypto extension.