

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

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

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

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

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

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 08-02-24 | Sean Chase | <Brief description of changes in this revision> |

**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.35nkun2)

The software design problem is to develop a game application in a web-based distributed environment. The solution is to create a Java application and use object-oriented programming principles. This application would need to include classes that will interact with one another for the game to be played. The classes are GameService, Game, Team, and Player. Together, they will manage the game, teams, and players. There will be an Entity class, which will be the base for all the entities with attributes like Id and name. Using this design approach will fulfill the software requirements. It will also make a functional game application.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_heading=h.1ksv4uv)

1. Web-based Distributed Environment: This constraint means that the application needs to be accessible over the internet and it should also support multiple concurrent users. It should also be able to handle network latency and security considerations.
2. Java Programming Language: The software needs to be developed using Java. This constraint means it is limited to Java based frameworks, libraries, and tools. It also means adhering to Java coding conventions and best practices.
3. Object-Oriented Design: The application needs to follow object-oriented design principles to promote modularity, maintainability, and reusability. This means that it should design classes with proper encapsulation, inheritance, and polymorphism. Also, it involves applying design patterns and abstraction techniques. These create a flexible and extensible codebase.
4. Scalability and Performance: The game application needs to be designed to support a growing number of games, teams, and players. It needs to be scalable so it can handle an increasing user base without causing significant performance degradation. This constraint requires optimizing database access. Also, minimizing resource consumption. And caching mechanisms should be considered.
5. User Interface: This application needs to have an intuitive user-friendly interface. It should be responsive. It needs to provide an engaging experience for the players, allowing them to interact with games, teams, and other players. This means the design should be a responsive and visually appealing user-interface using HTML, CSS, and JavaScript.

## [System Architecture View](#_heading=h.44sinio)

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.2jxsxqh)

The Gaming Room UML Diagram consists of several classes. They include GameService, Game, Team and Players. They are interconnected, fulfilling the software requirements. The Entity class serves as the base for all of the entities in the program. It provides common attributes like Id and name. This diagram shows the principle of inheritance. It allows other classes to inherit the attributes and behaviors. Using the Entity class means code duplication is avoided and it ensures consistency with the entities. The GameService class manages the games. It maintains the list of games and has methods to add games, retrieve games by Id or name, and also retrieve the total game count. This class has the singleton design pattern to ensure a single instance is accessible throughout the application. The SingletonTester class uses the ProgramDriver class to test the singleton behavior. The game class is for a game in the application. This maintains a list of teams participating in the game, and has methods to add teams and to retrieve a string representation of the team. The Player class represents a player and has a method to retrieve a string representation. These classes use the principle of composition. The Game class consists of teams, and the teams have players.

**"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.z337ya)

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** | It can be used as a server for web-based applications. It has a Unix-based environment which is good for hosting. It provides robust security and stability. It has Apache web server and PHP support. But there are limited hardware choices and it may be more expensive. | Popular choice for hosting web-based software. Offers range of distributions for server use. Good scalability and stability. It is cost-effective, offers wide range of server software and is very customizable. It requires more expertise to use. | This is usually used for hosting web applications, especially ones using Microsoft. It works well with Microsoft and offers many web server options. But licensing costs can be high and isn’t good for open-source software development. | These use servers to serve mobile data. Any platform can be used for mobile backends.Cost efficient and good for small-scale applications. But can have performance issues. Could have battery drain and hardware failures.Also maintenance and management complexities. |
| **Client Side** | Usually involves using Apple’s development tools. Cost is moderate. Objective-C and Swift expertise might be needed. | Expertise is required for distribution specific considerations. But the costs are usually low. | This usually involves using Visual Studio. Expertise in C# might be required, and costs vary. | Costs vary depending on number of platforms. Need expertise in Swift,or Kotlin. |
| **Development Tools** | Xcode is the primary integrated development environment. I supports Objective-C, Swift, and C+. | Linux supports many programming languages. Uses Visual Studio, and Eclipse. | Visual Studio is the primary IDE. It supports C#,C++, and many more. | For Android, Android Studio and Kotlin. For IOS, Xcode and Swift, Objective-C are used. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>