

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_gjdgxs)

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_2s8eyo1)

[**System Architecture View 3**](#_17dp8vu)

[**Domain Model 3**](#_3rdcrjn)

[**Evaluation 4**](#_26in1rg)

[**Recommendations 5**](#_35nkun2)

## [Document Revision History](#_1ksv4uv)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 02/27/2024 | Jarrod Helmers | Added information for software design |

**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](#_44sinio)

The new client, The Gaming Room, wants to extend their Android game from an Android app to a web-based platform, allowing multi-platform accessibility. The game is called Draw It or Lose It, inspired by the 1980s TV game Win, lose, or draw, and the challenge is to develop a web-based game application supporting multiple players. The goal is to ensure uniqueness for team names, enabling users to check name availability during team creation and only one instance of the game should exist at any given time. To achieve this, we propose a design using the singleton pattern for managing instances and the iterator pattern to validate and maintain unique names.

## Requirements

* *The game must support multiple teams and players*
* *Game and team names must be unique*
* *Only one instance of the games should exist in memory*

## [Design Constraints](#_2jxsxqh)

* Web-based distributed environment - The game is intended for web platforms, the design must consider the challenges of communications between distributed components.
* Singleton Pattern - We propose employing the singleton pattern to fulfill only one instance of the game in memory.
* Unique names - Ensuring unique game and team names pose a constraint on data validation. This would require a systematic approach to verify name availability.

## [System Architecture View](#_z337ya)

Please note: Nothing is 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](#_3j2qqm3)

The UML class diagram represents the domain model for the game application. The diagram includes the Entity class that acts as a base class holding common attributes, the Game class that inherits from the Entity class, and the Team and Player classes that also inherit from the Entity class, sharing a set of attributes and behaviors.

**"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](#_1y810tw)

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** | Mac is stable and Unix based, and suitable for macOS development but is proprietary and has higher costs | Linux is stable and widely used, cost-effective, and supports various server-side technologies but it requires expertise | Windows is user friendly, and supports .NET, ideal for Microsoft technologies, but also has higher licensing costs | Mobile devices require backend servers, they have access to a massive user base but development may be platform-specific |
| **Client Side** | Focus on user experience and macOS design guidelines with higher costs | Compatibility testing across distributions, lower cost, and expertise required | Address different Windows versions, UI guidelines, and specific tools | Separate development for iOS and Android. Higher costs, and separate development teams |
| **Development Tools** | Swift, Objective-C, Xcode | C, C++, Python, Java, Eclipse, Visual Studio | C#, C++, .NET, Visual Studio | Swift, Objective-C, Java, Kotlin, Xcode, Android Studio, React |

## 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**: A cross-platform development to ensure compatibility with various operating systems, which will allow for expansion. You will also want a platform that offers flexibility, and scalability.
2. **Operating Systems Architectures**: A development framework that abstracts underlying operating system complexities, and ensures consistency between different systems. Both Azure and AWS offer a variety of architectures that would provide flexibility and accommodate the needs of the client.
3. **Storage Management**: Cloud-based storage such as AWS offers scalability and accessibility, accommodating dynamic data requirements.
4. **Memory Management**: Employ cross-platform memory management practices and ensure the best practices for optimal performance across devices.
5. **Distributed Systems and Networks**: Develop a RESTful API to facilitate communication between platforms. Utilize cloud services such as AWS for scalable and reliable interaction. Consider potential dependencies and degradation during connectivity issues.
6. **Security**: Implement end-to-end encryption for data transmitted between platforms. Best security practices and regular updates and patches should be followed to address security vulnerabilities.