

<Name-of-Software-Application>

# **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 | <5/25/2025> | <Ariba Tariq> | Initial D raft |

**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)

The Gaming Room is excited to take its Android game, Draw It or Lose It, and transform it into a web-based, multi-platform experience. The plan is to create a scalable and maintainable architecture that sticks to essential software design principles while making good use of established design patterns. Some key objectives include making sure that only one instance of the game is running at any given time, giving each game, team, and player a unique identifier, and preventing the creation of duplicate teams or games. To make this happen, we’ll use the Singleton design pattern to manage the game’s lifecycle, and the Iterator pattern will help us efficiently create and navigate through teams and players.

## Requirements

*- Support multiple teams, each with several players.*

*- Enforce unique names for games and teams.*

*- Keep a single active game instance in memory.*

*- Generate unique identifiers for all games, teams, and players.*

*- Implement thread-safe mechanisms for handling user actions simultaneously.*

*- Ensure consistent performance across different operating systems and devices.*

## [Design Constraints](#_2et92p0)

- Only one game instance can be active at any given time.

- Uniqueness must be maintained for all game elements.

- The system should safely manage concurrent access.

- Cross-platform compatibility is a must.

## [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)

An abstract base class called Entity provides shared attributes like ID and name. The Game, Team, and Player classes inherit from Entity, which helps in reusing code effectively. The GameService, designed as a Singleton, oversees the creation and uniqueness of games, teams, and players.

The Iterator pattern is used to simplify the traversal and validation of collections related to games. This structure highlights essential object-oriented programming (OOP) principles. Inheritance is utilized through the base Entity class, allowing for code reuse among Game, Team, and Player classes. Encapsulation is achieved by centralizing the game management logic within the GameService class. The Singleton pattern guarantees that only one instance of GameService exists throughout the application’s lifecycle, while the Iterator pattern facilitates efficient navigation and management of collections, such as teams and players, especially for ensuring uniqueness and retrieving game data.**"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** | Mac servers create a reliable environment for hosting web applications, thanks to their Unix-based architecture that boosts security. However, they aren't as widely used, and the software support available can be somewhat limited. | Linux has become a go-to option for web hosting because of its impressive stability, strong security features, and the fact that it's cost-effective aas an open-source platform. It accommodates a broad range of server technologies, but setting it up and maintaining it usually demands a bit more technical know-how. | Windows servers offer a user-friendly management experience, equipped with tools like Internet Information Services (IIS), and they work seamlessly with other Microsoft products. They're particularly well-suited for applications built on the .NET framework, although the licensing costs can be a drawback. | While mobile devices aren't typically used to host web applications, they mainly serve as endpoints for accessing them. That said, they can run lightweight local servers for testing or specific scenarios involving specialized applications |
| **Client Side** | developing for macOS, you need to make sure everything works smoothly within Apple's ecosystem. This can lead to higher development costs and requires a specific skill set tailored to macOS. It's crucial to test your application on various versions of macOS to guarantee it runs well across the board. | On the other hand, developing for Linux on the client side is often simpler and gives you access to some powerful development tools. However, it can get tricky due to compatibility issues that arise with different distributions. To navigate this landscape effectively, you really need to have a solid grasp of Linux environments. | Windows is a popular choice for client-side development, thanks to its widespread use and the ease of finding development tools. It generally provides good compatibility for most web applications, but you have to ensure that your app supports a variety of Windows versions. | When it comes to mobile development, you're looking at creating apps for multiple operating systems like iOS and Android, as well as different device types. This process can be quite time-consuming, especially if you're building cross-platform applications or fine-tuning your app for various hardware configurations. |
| **Development Tools** | Developers working on macOS typically turn to Xcode for building applications, while tools like Visual Studio Code and IntelliJ cater to a broader range of development tasks. Popular programming languages in this environment include Swift, Objective-C, and JavaScript, especially for web applications. | On the Linux side, programmers have a wealth of versatile tools at their disposal, such as Vim, Emacs, VS Code, and Eclipse. The go-to programming languages here are Python, Java, C++, and JavaScript, with Docker and Git being widely used tools. | For Windows development, tools like Visual Studio, PowerShell, and VS Code are essential. C#, .NET, Java, and Python are among the favored programming languages. Windows also excels in supporting enterprise development and maintaining legacy systems. | When it comes to mobile development, the platforms differ quite a bit: Android developers often rely on Android Studio with Java or Kotlin, while iOS developers stick with Xcode and Swift. For cross-platform development, tools like Flutter, React Native, or Xamarin come in handy. |

## 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**: When it comes to choosing an operating platform, I highly recommend Linux. It's known for its stability, security, and cost-effectiveness, making it a solid choice.
2. **Operating Systems Architectures**: For the operating systems architecture, consider implementing a microservices approach. This will allow you to manage game logic, team management, and player interactions independently, which can really streamline the process.
3. **Storage Management**: In terms of storage management, a relational database like MySQL is a great option for keeping track of game, team, and player data.
4. **Memory Management**: For memory management, leveraging Java’s built-in garbage collection will help you handle memory allocation and cleanup efficiently, so you can focus on other important tasks.
5. **Distributed Systems and Networks**: When it comes to distributed systems and networks, using RESTful APIs is key. They facilitate smooth communication between platforms, ensuring everything is compatible and scalable.
6. **Security**: Always use HTTPS for data transmission, implement authentication tokens to verify users, and make sure to apply input validation to prevent any injection attacks<.