

The Gaming Room

# **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 | 08/24/2024 | Jessica Chaput | creating The Gaming Room application specifications |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to expand its existing game, “Draw it or Lose it,” an Android app, to a web-based game that supports multiple platforms. The solution to this is developing software that supports web-based and multiplatform access while ensuring unique game instances and managing multiple teams and players.

## Requirements

Business Requirements:

Expand Platform – expand existing Android game “Draw it or Lost it” to be web-based and accessible for multiple platforms.

Support Multiple Teams – The game should support multiple teams, with each team having multiple players.

Unique usernames: Both game and team names must be unique, allowing users to quickly identify if a name is already in use.

Single Instance for games and team names: only one instance of the game should exist in memory to maintain consistency and prevent errors.

Technical Requirements:

Web-based Development: The application must be developed to function as a web-based service, accessible through different browsers and operating systems.

Singleton – This application should be implemented as a singleton.

Unique identifiers: Unique identifiers must be generated for each game, team and plater to ensure data integrity and prevent conflicts.

Scalability: The application needs to be scalable to accommodate an increasing number of users.

## [Design Constraints](#_2et92p0)

Independent Platforms:

Constraint – The application must be accessible to multiple platforms and operating systems via a web-based interface.

Implications: This requires the use of platform-independent technologies such as HTML, CSS, or JavaScript for front-end development and Java for back-end development. Because of this constraint, the application should also be tested on all browsers and devices to ensure proper compatibility.

Scalability –

Constraint – The system must be able to scale to support a growing number of users, teams, and games.

Implication: The application architecture must be designed to support horizontal scaling, potentially using cloud-based infrastructure.

Singleton Pattern:

Constraint – Only one instance of the game service should exist in memory at any time to ensure consistency and prevent conflicts.

Implication: The use of the Singleton design pattern is mandatory for the game service class. This restricts the instantiation of the service to a single object and ensures that all parts of the application interact with the same instance.

Unique identifiers:

Constraint: Each game, team, and player must have a unique identifier.

Implications - The system must implement a robust mechanism for generating and managing unique identifiers

Security:

Constraint – The application must secure user data

Implication - The application must implement security protocols for data, such as encryption and user authentication.

## [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 UML class diagram for The Gaming Room project outlines the relationships and structure of the key classes involved in the application. The classes in the UML diagram include:

**Entity Class**

The entity class has attributes:

-Id: long

-Name: String

This base class provides common attributes (id and name) to other classes. It serves as a parent class for Game, Team, and Player.

**Game Class:**

The Game class has attributes:

-Teams: List<Team>

This class shows inheritance and inherits from the Entity class. The Game class represents a single game instance. It has a one-to-many relationship with the Team class, indicating that a game can have multiple teams. The class included methods to add teams and manage them within the game.

**Team Class:**

The Team class has attributes:

-players: List<Player>

This class also inherits from the Entity class

The Team class represents a team within a game. It has a one-to-many relationship with the Player class, indicating that a team can have multiple players. The class included methods to add players and manage them within the team.

**Player Class**

The Player class has no attributes.

This class inherits from the Entity class as well. The Player class represents an individual player within a team. This class is the simplest in the hierarchy, inheriting the id and name attributes from the Entity class.

**GameService Class:**

The GameService class has attributes:

-Games:List<Game>

-nextGameId: long

-nextPlayerId: long

-nextTeamId: long

The GameService class is responsible for managing game instances. It follows the Singleton design pattern, ensuring only one instance of GameService exists at any time. This class manages the creation and retrieval of games, teams, and players. It also ensures unique identifiers.

**ProgramDriver Class:**

This class has our main method

+ main()

The ProgramDriver class serves as the entry point of the application. It initialized the GameService singleton and managed the overall flow of the program.

**SingletonTester Class:**

this class contains the method

+ testingSingleton()

The SingletonTester class is used to verify the correct implementation and behavior of the Singleton pattern in the GameService class.

Overall, this UML diagram demonstrates key object-oriented programming principles such as inheritance, where the Entity class serves as a base for the Game, Team, and Player classes. Another method shown is encapsulation. Encapsulation is employed by using private fields with public getters, protecting the internal state of objects. The Singleton pattern is also implemented, specifically in the GameService class. The singleton pattern ensures only one instance of identifiers exists at any time.

**"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

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS provides a stable and secure environment for hosting web-based applications, largely due to its Unix-based architecture. It integrates well with Apple’s ecosystem, making it a good choice for developers already invested in Apple’s tools.  Pros:  Stability and security are strong points, and it's developer-friendly with tools like Xcode. Mac hardware is typically high-performance, which can be beneficial for hosting.  Cons:  The hardware and licensing costs can be higher than other platforms, and macOS is less customizable, which could be a drawback for some server configurations. | Linux is well-regarded for its stability, security, and flexibility, making it a popular choice for hosting web applications. It supports a wide array of open-source tools and can be run on various hardware configurations, offering cost efficiency.  Pros:  Open-source, cost-effective, highly customizable, and supports a wide range of programming languages and development tools. It’s widely used in cloud environments and easily scalable.  Cons:  Requires more technical expertise to set up and manage. Support may be community-based, which can be less reliable than commercial support options. | Windows Server is widely used in enterprises and offers broad support for Microsoft technologies. It is known for its user-friendly interface and strong support network.  Pros:  Extensive support for enterprise applications, including .NET and SQL Server. It’s easy to integrate with other Microsoft products and services, making it a good option for companies already using Microsoft software.  Cons:  Higher licensing costs and the need for regular updates and reboots can be disruptive. Windows is also generally considered to be less secure than Linux. | Mobile devices are typically not used as servers but can host lightweight services or applications for local use.  Pros:  Portable and increasingly powerful, mobile devices can be used for specific, low-power tasks or as part of edge computing setups.  Cons:  Limited by hardware capabilities and not suitable for full-scale web server deployment. Reliability and performance may not meet the demands of a web-based server environment. |
| **Client Side** | Developing for macOS typically requires knowledge of Swift and Objective-C, using Xcode as the primary development environment.  Pros:  High-quality development tools and a strong emphasis on user experience and design. macOS applications are known for their polished interfaces.  Cons:  Development costs can be high due to the need for Apple hardware and licensing. | Linux client development often involves multiple distributions, each with unique requirements. Popular languages include Python, C++, and Java.  Pros:  Low development costs with many free and open-source tools available. Linux is highly customizable and can be tailored to specific needs.  Cons:  Fragmentation across distributions can lead to compatibility issues. Less uniformity in user experience compared to macOS or Windows. | Windows supports a wide range of development tools and languages, including .NET, C#, and Visual Studio.  Pros:  Strong support network and a broad user base. Visual Studio is a powerful IDE with extensive features.  Cons:  Licensing costs can add up, and Windows applications may require more maintenance due to frequent updates. | Mobile development varies by platform—iOS uses Swift/Objective-C with Xcode, and Android uses Java/Kotlin with Android Studio.  Pros:  Both platforms have strong development ecosystems with robust tools. Cross-platform frameworks like React Native can also be used to develop for both iOS and Android.  Cons:  Development can be complex due to the need to support multiple devices and operating systems. Licensing and device testing can increase costs. |
| **Development Tools** | macOS development tools include Xcode, which supports Swift and Objective-C. Xcode is comprehensive and integrates well with other Apple technologies.  Pros:  Streamlined development process for macOS and iOS applications. High-quality tools and documentation.  Cons:  Requires Apple hardware, which can be expensive. | Linux offers a broad range of development tools, including GCC, Make, Eclipse, and text editors like Vim and Emacs.  Pros:  Most tools are open-source and free, making Linux a cost-effective development environment. Supports multiple languages and frameworks.  Cons:  Requires more technical expertise to set up and use effectively. Some tools may have a steeper learning curve. | Windows development leverages tools like Visual Studio and the .NET framework, with languages such as C# and C++.  Pros:  Visual Studio is a powerful IDE with extensive support for various applications. Strong integration with Microsoft services.  Cons:  Licensing fees for Visual Studio and Windows can be high. | iOS development uses Xcode with Swift and Objective-C, while Android development uses Android Studio with Java or Kotlin.  Pros:  Both platforms have comprehensive development environments with strong support networks.  Cons:  Development costs can be higher due to licensing and the need for device testing. Cross-platform development tools can add complexity. |

## 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**: I recommend using Linux as the primary operating system to expand "Draw It or Lose It" to other platforms. Linux is known for being reliable, flexible, and accessible, which makes it an excellent choice for The Gaming Room's needs. It's perfect for web-based applications, which we need as we expand the game to work on multiple platforms. Plus, Linux works well with cloud services like AWS or Google Cloud, making it easier for players to access the game from anywhere.
2. **Operating Systems Architectures**: Linux offers different ways to set up the system depending on our needs. One of the best things about Linux is that it supports tools like Docker, which allows other parts of the game to run separately but still work together. This is important for keeping the game running smoothly, especially when many people are playing simultaneously. We can ensure the game doesn't slow down or crash by spreading the workload across different containers.
3. **Storage Management:** I suggest using cloud storage like Amazon S3 or Google Cloud Storage to store all the game data. These services are reliable and can handle much data, such as user profiles and game progress. They also automatically back up the data, so we don't have to worry about losing anything. Plus, if something goes wrong, we can quickly restore the data.
4. **Memory Management**: Linux is great at managing memory, which is essential for running the game well. It uses virtual memory, which lets the game use part of the computer's storage as extra memory. This is especially useful when the game needs to handle more data than the computer's physical memory can. Linux also ensures that only the necessary parts of the game are loaded into memory, which helps keep everything running smoothly.
5. **Distributed Systems and Networks**: To ensure "Draw It or Lose It" works well on different platforms, I recommend breaking the game down into smaller parts, microservices. These parts can communicate using APIs, allowing them to work together even if they're running on different systems. For example, one part could handle user logins, while another handles the game logic. To ensure the game runs fast no matter where the players are, we can use Content Delivery Networks (CDNs) to distribute game files across servers worldwide. We can set up backup systems to keep the game online if there are network issues.
6. **Security**: Security is a big deal, so we must protect user data. On Linux, we can use firewalls to control who can access the server and encryption to protect data stored or sent over the internet. Using multi-factor authentication is also a good idea to ensure that only authorized users can access the system. Regularly updating the server with security patches will help protect against new threats. Finally, setting up specific permissions for users and processes will minimize the risk of accidental or malicious actions that could harm the system.