

Draw It Or Lose It

# **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 | July 21, 2024 | <Brandon Parkerson> | First Production |

**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 Gaming Room has been determined to have a web-based version of their game called “Draw It or Lose It”. The game’s capability should allow a multiple number of teams consisting of multiple players per team. Each game, team, and player instance should only happen one. Singleton creation pattern is utilized for object creation to block multiple game instances and an iterator pattern to prevent conflicting teams and members.

## 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)

The Gaming Room already has a mobile version of Draw It or Lose It. They have now asked to push their product to the web. The tech stack must be working with a web environment for this project. Java, being the original language, should help in this new extension of the game.

## [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. 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.**

The program features a main driver class responsible for initializing the creation of games, teams, and players. This creation process is managed by the GameService class, which adheres to the singleton design pattern to ensure only one instance of GameService exists in memory at any given time.

GameService prevents the creation of multiple instances by making its constructor private. The only way to obtain an instance of GameService is through the getInstance() method. This method checks if an instance already exists and creates one only if it does not.

Once the GameService instance is active, the driver class can invoke the addGame() method. This method employs the iterator pattern to avoid creating Game objects with duplicate names. Newly created Game objects are stored in the games list.

After a game is established, teams can be added using the addTeam() method. Similar to addGame(), addTeam() uses the iterator pattern to prevent the addition of duplicate Team objects to the game. New Team objects are then added to the teams list.

Following the creation of teams, players can be added to teams through the addPlayer() method. This method also utilizes the iterator pattern to avoid duplicate Player objects within a team. New Player objects are subsequently added to the players list.

The designed UML demonstrates several object-oriented programming techniques. Polymorphism and inheritance are evident in the Entity class extension and constructor overloading. Encapsulation and abstraction are showcased in the method for adding teams. Direct creation of Team objects is not possible due to the protected constructor; however, teams can still be added via the addTeam() method, abstracting the underlying creation process from the user.

The Game, Team, and Player classes all inherit from the Entity class. Entity features two protected attributes: id and name. Its default constructor is protected to prevent the creation of null objects, ensuring only the overloaded constructors are used.

## [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** | Macs are usually not used for hosting web-based applications because they’re more expensive and not as common as Linux servers. They have good security and work well with other Apple products, but Linux is generally preferred for hosting due to its lower cost and broader community support. | Linux is a popular choice for web servers because it’s free, flexible, and has a lot of community support. It’s cost-effective and runs well, making it a solid option for hosting websites and applications. | Windows is a popular choice for web hosting, largely because of its extensive user base and the range of developer tools it supports. Its robust scalability and performance capabilities make it a strong option for handling large volumes of traffic and complex applications. Additionally, its integration with other Microsoft products can streamline development and management tasks. However, it’s worth noting that the licensing costs for Windows Server can be significant, so it’s crucial to budget accordingly and consider whether the benefits align with your specific needs and resources.   |  | | --- |  |  | | --- | | Mobile devices aren’t used to host web-based applications because they don’t have enough power or storage. They’re meant to access apps hosted elsewhere. The focus here is on making sure apps work well when accessed from mobile devices like smartphones and tablets. |
| **Client Side** | | Developing apps for Macs can be expensive because Apple hardware costs more. You’ll need to use Xcode for development, and know Swift or Objective-C. It’s important to think about how your app will look and work on Macs compared to other platforms. | | --- |  |  | | --- | | | Developing for Linux is cheaper because many tools are open-source. But, Linux comes in many versions, so making sure your app works across all of them can be tricky. You might need to use languages like Python or Java. | | --- |  |  | | --- | | | Windows is a major platform for apps, so supporting it is important. Development costs can be reasonable, and you can use tools like Visual Studio. Knowing languages such as C# and .NET is helpful, and since many people use Windows, it can reduce support and training needs. | | --- |  |  | | --- | | Developing for mobile devices means using Swift or Objective-C for iOS and Java or Kotlin for Android. Tools like Flutter and React Native can help you build apps that work on both platforms. It’s important to make sure your app works well on different devices and screen sizes. |
| **Development Tools** | | On Macs, you’ll mainly use Xcode for developing apps, especially if you’re making apps for macOS or iOS. You’ll work with Swift or Objective-C. For web development, tools like Homebrew and Node.js can be helpful. Macs also work with other development tools if you’re doing cross-platform work. | | --- |  |  | | --- | | | On Linux, you can use many free and open-source tools. Languages like Python, C++, and Java are common, and you can use IDEs like VS Code and Eclipse. Linux is also great for server-side work with tools like Apache and Docker. | | --- |  |  | | --- | | | Windows development usually involves using Visual Studio, which is a powerful IDE for languages like C# and C++. Windows also supports other tools such as Eclipse and IntelliJ IDEA. It’s a versatile platform for both desktop and web development. | | --- |  |  | | --- | | For mobile development, you’ll use Xcode for iOS apps and Android Studio for Android apps. Tools like Flutter, React Native, and Xamarin are useful for creating apps that work on both iOS and Android. You’ll work with languages such as Swift, Objective-C, Java, Kotlin, and Dart. |

## 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**: For expanding "Draw It or Lose It" to various computing environments, Linux is an excellent choice. Linux is widely used in server environments and can handle a range of applications from web servers to complex databases.
2. **Operating Systems Architectures**: Linux operates on a monolithic kernel architecture, meaning the kernel manages the system's hardware and executes core functions such as process management, memory management, and device drivers all within the kernel space. This architecture allows for efficient performance and faster execution.
3. **Storage Management**: For storage solutions, we strongly recommend considering Microsoft Azure due to its cost-effectiveness, exceptional customer support, and commitment to regular updates and improvements.
4. **Memory Management**: Windows 11 is the latest version of the Windows operating system and continues to advance its memory management techniques for enhanced performance and efficiency. It utilizes disk paging and demand paging to optimize memory usage. Disk paging involves allocating a portion of the hard drive to function as additional virtual memory, effectively extending the system's physical RAM. Demand paging improves efficiency by breaking down processes into smaller segments, which are only loaded into memory when needed for immediate execution. Additionally, Windows 11 offers a comprehensive virtual memory address space, providing ample resources for handling applications and their memory demands.
5. **Distributed Systems and Networks**: We suggest choosing Azure as your cloud service provider because it simplifies managing distributed systems and networks. Azure provides excellent uptime and offers helpful tools like cloud-based email alerts, Azure App Insights for logging, and comprehensive monitoring solutions. These features are especially important as you scale up to support up to 1,000 concurrent games, each with 4 players. By leveraging Azure to handle network demands, you can concentrate more on enhancing the application itself and its features.
6. **Security**: Azure simplifies the management of security for user information and personal data. Devices will connect to the "Azure App Service" through an App Service Plan, utilizing Azure Active Directory for secure logins. Azure also offers several advanced security features, such as:

IP configuration options to whitelist access to specific resources, like player or personal data, or to restrict access to the entire application.

The ability to store data within a Virtual Private Network (VPN) in the cloud for enhanced protection.

Database access restricted to IP whitelisting, password protection, and SSL connectivity to safeguard user information.