

Draw it or Lose it

# **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 | 07/17/24 | Elizabeth Wagner | Includes multi-player feature requirements for the web-based game Draw it or Lose it. |

**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 wants to design a web-based version of Draw it or Lose that will make the game available on multiple platforms to attract new users, with help from Creative Technology Solutions (CTS). They want to implement features for multiplayer team support, unique player/team identifiers, and single instance control for each game, team, or player. These features will help improve gameplay and user interaction, while creating a seamless cross-platform experience.

## Requirements

* *The game must be redeveloped as a web-based game.*
* *The game must support one or more teams that can contain multiple players.*
* *Unique Identifiers for game, team, and player names to prevent issues during setup or gameplay.*
* *The system must enforce single-instance control to only allow one active instance of any game, team or player at one time.*

## [Design Constraints](#_2et92p0)

* Must run on web-based platforms which may result in issues with security, network communication, and compatibility with different browsers or devices.
* The system needs to ensure that game, team, and player names are unique, requiring it to check if any names are already being used.
* There must only be one active instance of a game, team or player at one time to reduce the risk of data conflicts or performance issues.

## [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 diagram below shows core object-oriented principles, specifically inheritance, encapsulation, and abstraction. The Game, team, and player classes extend the Entity class, inheriting shared attributes such as id or name, which helps with code reuse as well as consistency. The GameService class demonstrates encapsulation by keeping its internal data private and allowing controlled access through public methods. It also uses the singleton pattern to help make sure a single instance manages all games, meeting the requirements for single-instance control. Abstraction is used to simplify the interactions between games, teams, and players by focusing on the important features while hiding the more complex background processes.

**"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 as a server platform can be dependable and is a developer-friendly environment especially for those already familiar with Apple’s programs. It has a polished, easy to navigate interface and runs smoothly. Although it may not be the best option for hosting large-scale web applications due to limited hardware options and potential licensing costs, especially when trying to expand server resources to handle a growing user base. | Linux is a great choice for hosting web-based applications because of its performance, flexibility, and is cost efficient. Since it’s an open-source operating system, it can easily be tailored to meet server specific needs as well as performing well and managing heavy server loads. It is also very compatible with major web tech like python, PHP, java, ruby, and many others while also having a lot of resources to help with everything from trouble shooting to performance optimization. | Windows servers are a strong option for development teams familiar with the Microsoft operating system. The GUI helps make server management easier to understand, especially for those with less experience. Windows servers integrate seamlessly with platforms such as SQL, Office, and Azure, which make managing applications in hybrid environments easier. It also supports tech like .NET, ASP, and Power Shell, which provides more flexibility with development. However, compared to Linux, the licensing and maintenance costs can be significantly higher, which can impact budgeting and large-scale applications. | Mobile devices are not typically used to host web-based applications, but they do play a big role in how users can access web content. It’s important to make sure the application runs smoothly on various smartphones and tablets to provide a reliable and enjoyable experience whether it’s on Android or iOS. Since most users rely on mobile devices for everyday tasks, optimizing performance and usability on smaller screens is vital. Focusing on mobile support helps the application stay competitive and accessible in today’s mobile-driven environment. |
| **Client Side** | Developing for Mac usually requires using tools like Xcode and knowledge of programming language such as swift or Objective-C. This can potentially take more time and may cost more if your development team isn’t already familiar with Apple’s system. Macs are known for being stable and having a great user interface, but since less people use Mac compared to other operating systems, it may not be a top priority unless most of the users are using apple products. | Supporting Linux on the client side can be a little tricky due to there being many versions and desktop setups. This means the developers may need to spend some extra time to make sure the app works the way it should. The good news is most development tools for Linux are free and open source, which will help with reducing costs. Developers who have knowledge in languages such as C++, python, or java and are also comfortable with using Linux tools will have an easier time building for this platform. | Windows is the most common desktop system globally. Many developers already know how to work with Windows, using tools like Visual Studio which will help save some time and money. Windows also supports a large range of hardware and software, which will help it run smoothly on most devices. However, developers should account for potential costs of licensing certain tools or features. Regular updates and system changes may also impact testing and compatibility. | Supporting mobile devices will mean building apps for both Android and iOS. This could be more time-consuming because each platform uses different tools and programming languages. It may also lead to higher development costs if separate teams or skillsets are needed. Cross-platform frameworks like flutter and React Native could be used to reduce time and effort by allowing developers to create shared code for both systems. Regardless of the method, it will be important to run testing on different screen sizes and device hardware to make sure the app performs well for all users. |
| **Development Tools** | Mac mostly uses Swift or Objective-C, with Xcode as the primary IDE. Xcode includes all the tools needed to write, test, and debug applications that are being built for macOS. Developers can also use tools like Flutter or Electron to help create apps that run on multiple systems, while also running smoothly on Mac. | Linux developers often use languages such as Python, C++, and Java, with popular IDEs like, Visual Studio, Eclipse, and JetBrains products (ex. PyCharm, IntelliJ IDEA, CLion, etc.). Development on Linux usually relies on command-line tools, package managers and build systems like CMake to make the compilation process smoother. The flexibility and variety of tools make Linux well-suited for specialized and large-scale development of applications. | Windows development commonly uses C#, .NET, and C++ along with Visual Studio which is the main IDE for creating applications. Many developers also use Java, Python, or Node.js for cross-platform solutions. PowerShell also allows for powerful automation capability. In addition, the Windows subsystem for Linux (WSL) allows developers to use tools from Windows and Linux in the same environment. | Mobile application development relies on different things depending on whether its iOS or Android. For iOS, it’s Swift or Objective-C with Xcode as the main IDE. For Android, it’s Java or Kotlin, with Android studio as the primary IDE. Cross-platform frameworks like Flutter, React Native or Xamarin will allow developers to build apps that will work on both operating systems. Both IDEs have built-in tools to help make it easier for testing and refining applications for different devices. |

## 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**: Linux would be the most suitable operating platform for not only hosting but also expanding due to its flexibility, performance, and cost-effectiveness. Since it’s an open-source system it’s both affordable and very customizable, which is perfect for building a large-scale web-based application. Linux is known for its great performance and stability under heavy workloads, ensuring reliability for the application as the user base grows. Linux is also compatible with a wide range of programming languages and frameworks, making it easier for development and integration.
2. **Operating Systems Architectures**: Linux uses a layered design that helps make it stable, secure and flexible for hosting applications like Draw It or Lose It. The Linux kernel, acting as the core of the operating system, controls important system resources like memory, processors, and storage devices as well as handling hardware communication. The next layer consists of system libraries and tools that allow the developers and applications to easily interact with the kernel without having to directly access hardware. In the top layer is where the user applications like servers, databases, and development software are located, which can be customized or replaced as needed. This structure also allows new features or services to be added without affecting the rest of the system.
3. **Storage Management**: Linux offers a few different options for managing storage that balances performance, flexibility, and reliability. One option is the ext4 file system, which is usually chosen as default, known for its stability and compatibility. Another is XFS, known for its efficiency in handling large files and high-demand workloads. There is also Btrfs, which also includes some advanced features like snapshots, data compression, and built-in data protection. To make storage more flexible, the Logical Volume Manager (LVM) allows storage to be resized and reallocated without having to take the system offline. Combined with RAID support for redundancy and fault tolerance, Linux has options that can meet current storage management needs as well as future expansion needs for Draw It or Lose it.
4. **Memory Management**: Linux uses a system called virtual memory to manage memory in a fast and efficient way, even when handling a large workload. This system allows the program to use more memory than what is physically available by storing the data temporarily on disk as swap space. The Kernel controls allocation and recovery automatically, only giving the application the memory it needs and freeing up space when it’s no longer in use. Features like paging and caching improve speed by keeping frequently used information in memory and moving unnecessary data to disk. Additionally, Linux uses memory protection, which keeps processes isolated to prevent and reduce the risk of faulty or even malicious programs from corrupting others.
5. **Distributed Systems and Networks**: To support communication across multiple platforms, Draw It or Lose It should be designed as a distributed application. This means the data and main game logic are hosted on centralized servers, while client devices like web browsers, mobile apps, or desktops connect over a network to play. The communication between them can be handled using APIs or services such as REST or WebSockets, which will allow real time updates during gameplay. Hosting the application in a cloud environment or internet-connected network will allow greater accessibility and scalability, while redundancy features like load balancing or failover servers help maintain service availability if a part of the system goes down. To handle issues with connectivity, tools like caching and messaging queues can keep the game running until the connection is restored if the player temporarily loses connection. Data synchronization can also make sure that players always see the most current game state, even when switching devices. Security measures like encryption and authentication must be included to protect the users accounts and game data.
6. **Security**: Since players can access the game from different devices and platforms, security is vital. With Linux, strong protection can be accomplished by using built-in tools such as firewalls, SELinux, or AppArmor, alongside strict access control to limit unauthorized activity. Sensitive information, including user credentials and game data, should always be encrypted when stored. Databases can also be protected with role-based access controls. Data that travels between clients and servers must also be secured with strong encryption protocols like TLS or SSL, to prevent interception. User accounts should be protected with secure password storage practices, including salting and hashing, as well as multifactor authentication, which will provide an extra layer protection. Additionally, regular updates and security patches will keep the system protected from potential threats. This, along with monitoring and logging will help detect unusual or suspicious activity early. Applying least-privilege principles, securing APIs, and keeping up with regular backups, can help make sure the users’ data remains safe and can be quickly restored if needed.