

<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 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <9/21/24> | <Wayne Tucker> | <Brief description of changes in this revision> |
| 2.0 | 10/6/24 | <Wayne Tucker> | <Updated the evaluation section for the characteristics of operating platforms> |
| 3.0 | 10/19/24 | <Wayne Tucker> | <Updated the recommendation section > |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to further expand their game, *Draw It or Lose It*, into a web-based platform. The game, inspired by the 1980s game show *Win, Lose, or Draw*, involves teams competing to guess images being drawn within a limited time. The goal is to develop a web-based version of the game to support multiplayer functionality, team management, and real-time interaction.

## Requirements

This requires an accessible, scalable, and secure web-based solution that can be used by multiple users across different platforms. The game should also support multiple teams with individual players. Players, teams, and games should have unique identifiers, with only one instance of the game running at a time for efficient memory management. Players should be able to guess the drawings in real-time*.*

## [Design Constraints](#_2et92p0)

***Programming Language***: While Java is a strong choice for backend logic, JavaScript for client-side development in a web environment.

***Rendering Drawings***: The game requires image render updates every second for 30 seconds.

***Server Hosting***: A local server might limit scalability and will be expensive.

## [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 shows **a** parent class called Entity, which connects to the Player, Team, and Game classes. These child classes inherit common attributes from Entity, such as id and name, and each class adds its own functionality to it. The **GameService** class manages these entities, such as assigning unique IDs to players, teams, and games, while making sure that names are unique.

The **Singleton pattern** is used in the GameService class for only one instance of the service is running. The GameService handles the creation and retrieval of games, teams, and players, and it coordinates with the main program driver. The **ProgramDriver** uses GameService to initialize the game data and test the Singleton functionality through a **SingletonTester** class.

**OOP**:

* **Inheritance**, where Player, Team, and Game all inherit common traits from Entity, promoting reusability.
* **Encapsulation**, as the attributes id and name are kept private and accessed through getters.
* **Abstraction**, as Entity provides a generalized template for game-related objects, hiding unnecessary implementation details.
* **Polymorphism** allows objects like Player, Team, and Game to be treated as Entity objects.

**"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 OS is known for its strong security and performance, making it a reliable option for web-based hosting. However, it is not commonly used for large-scale server hosting because it has limited support for server software, and hardware costs are higher than Linux or Windows. However, the proprietary nature of macOS limits flexibility. Server infrastructure for Mac OS is more expensive to set up and maintain. It’s less commonly used for cloud-based solutions like AWS, which favor Linux environments. Also, Mac OS licenses are more expensive than Linux, and the hardware costs are considerably higher, which could increase the overall cost of deployment. | Linux is the most popular OS for web servers, with its open-source nature providing flexibility, security, and a wide range of server software support. Linux is the preferred OS for hosting on cloud services like AWS and Google Cloud. While Linux is powerful for server hosting, it requires skilled administrators with Linux to maintain and configure systems properly. This can increase employee costs if their experience is lacking. Linux is open-source, meaning it has no licensing costs for the OS itself. This makes it a cost-effective solution for server-side deployment. | Windows is widely used in corporate environments and supports a range of server solutions like Windows Server and cloud platforms like Microsoft Azure. Windows has familiar development tools and a large pool of available talent. However Windows licensing costs for server software can be high, especially with Microsoft’s Azure services. Some argue that Windows is less secure than Linux for hosting, although Windows Defender and other built-in tools offer strong protection. Windows Server comes with substantial licensing fees. Also, Cloud-based options such as Azure have flexible pricing models, but licensing and usage fees may still add up over time. | Mobile devices, including smartphones and tablets, are extremely accessible, but they are not suitable for hosting large-scale applications. Instead, mobile devices are primarily used for accessing server-hosted applications. However, Limited processing power and storage on mobile devices restrict their ability to function as servers. They rely heavily on remote servers for hosting applications. Also, There are no direct server-side licensing costs associated with mobile devices, as they are not used for hosting applications. |
| **Client Side** | macOS offers a stable and secure environment for client-side applications. Its ecosystem supports responsive web development through browsers like Safari, which are built on WebKit, a solid rendering engine. However, Development for macOS requires experience with Swift and Xcode, which adds time and cost to the development process. The user base is smaller compared to Windows, potentially limiting the client’s reach. Also, Swift and Xcode are required for native macOS development, and testing on macOS systems is needed to ensure compatibility. This result in higher development costs, especially if specialized developers are required. | Linux offers a wide range of IDEs and browsers, including Firefox and Chrome, which ensures compatibility with modern web standards. It is also highly customizable for different user needs. However Development for Linux client-side applications may require additional testing for compatibility with different distributions, which increases the complexity and time needed for development. Also, Development for Linux will require a deep understanding of open-source technologies and the various distributions. Testing for compatibility is mandatory, because of fragmentation form the distributions. | Windows has the largest desktop user base, making it a key platform for client-side applications. Its development environments, like Visual Studio, are well-established and familiar to many developers. Development on Windows can be done using C#, .NET, and Java. However, the development process can be costly due to higher licensing fees for Windows operating systems and the development tools. Also, Client-side development on Windows benefits from tools like Visual Studio, which can reduce development time. However, licensing costs and developer may add to the project’s budget. | Mobile devices are ubiquitous, and supporting both iOS and Android expands the user base. Mobile development tools like Android Studio and Xcode allow developers to create applications for both platforms. However, Developing for multiple mobile platforms requires expertise in different environments such as, Kotlin for Android, Swift for iOS. Additionally, optimizing for smaller screens can be time-consuming and can require separate testing. Also, Using cross-platform frameworks like React Native can help reduce the cost and time of development by sharing code between platforms. |
| **Development Tools** | Key tools include Xcode, Swift, and Cocoa frameworks. Xcode is free, but hardware like Apple computers are required for development, which is costly. A specialized team of developers familiar with Apple’s ecosystem and languages would be necessary. This may require a separate development team, especially if macOS applications are developed alongside Linux or Windows versions. Also, Xcode and Swift are free, but the cost of macOS hardware can significantly increase the budget for the development environment. | Linux supports a variety of IDEs like Visual Studio Code, Eclipse, and Atom. It also supports a wide array of languages such as Python, JavaScript, and Java, making it a versatile development platform. Also, Linux development generally requires developers skilled in open-source tools and scripting. However, since Linux uses many cross-platform languages, the team does not need to be as specialized compared to macOS development. Most development tools and software libraries are open-source, reducing licensing costs to zero. | Visual Studio, .NET, and popular languages like C#, C++, and JavaScript are widely used in Windows development. These tools are integrated with Microsoft’s services, providing a powerful development environment. Windows development generally requires developers familiar with .NET, C#, and other Windows-native technologies. These are widely used, so finding developers should not be difficult. Also, Visual Studio has different licensing tiers, from free to paid, which can add to development costs depending on the version used. Windows licenses for client systems also need to be considered. | Android Studio for Android and Xcode for iOS are the main development environments. React Native and Flutter are popular for cross-platform development. Developers skilled in mobile platforms, along with those familiar with cross-platform frameworks, will be required. The project may need multiple teams or developers who specialize in each mobile platform. Android Studio is free, but Xcode requires an Apple Developer account for iOS app distribution. Cross-platform tools like React Native are free, though additional plugins or extensions may have costs. |

## 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 Windows as the operating platform for "Draw It or Lose It" due to its familiarity and wide usage among developers and users. Windows provides a strong development ecosystem, including tools like Visual Studio, making it easier for the development team to adapt and deploy across multiple computing environments. With support for cloud services like Amazon Web Services or AWS, Windows offers scalability, high availability, and easy cross-platform support for future growth.
2. **Operating Systems Architectures:** The Windows operating system architecture is well-suited for managing large-scale applications with its kernel mode and user mode components. In kernel mode, Windows handles core tasks like memory and process management, while user mode manages access to the file system, networking, and system APIs. This makes sure that "Draw It or Lose It" operates by distributing tasks between core system processes and user-driven actions.

:

1. **Storage Management**: For storage, I recommend leveraging Amazon S3 as a scalable, cloud-based storage solution. Amazon S3 offers cost-effective options for storing large image libraries, game data, and user profiles. Cloud storage solutions provide the benefit of allowing players across different regions to access game data quickly and reliably. While also, organizing data with metadata, that only relevant processes are loaded while reducing latency.
2. **Memory Management**: Windows supports advanced memory management techniques like virtual memory and paging, which are key to optimizing performance for "Draw It or Lose It." To enhance responsiveness, I recommend using techniques like lazy loading, where only necessary images and assets are loaded into memory when required. This prevents unnecessary memory usage and improves game speed. Windows also provides memory protection features, which prevent memory leaks and other common issues that could impact game performance over time. Additionally, memory compatibility across platforms whether the game is running natively or on virtual machines.
3. **Distributed Systems and Networks**: For "Draw It or Lose It" to function across multiple platforms, I recommend using Amazon Web Services (AWS) to implement a distributed system. AWS can easily handle the game’s server-side requirements, across platforms such as mobile, desktop, and web. By utilizing AWS Elastic Load Balancing and Amazon Route 53, the game can efficiently distribute traffic and maintain low latency, even during high traffic. Also, disaster recovery plans with automatic backups and redundancy can help during connectivity issues and data loss during outages.
4. **Security**: Security for The Gaming Room, and Windows provides several built-in features to for the user data to remain protected. Windows Defender and firewalls offers baseline protection, while AWS Security Hub provides advanced threat detection and mitigation against network security threats, including DDoS attacks. For secure communication between platforms, I recommend SSL/TLS encryption and multi-factor authentication can further protect user accounts.