

GamingRoom Software Design v1.0

# **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 | 3/25/2024 | Jon Guarante | First version of software design |
| 1.1 | 4/5/2024 | Jon Guarante | Added additional Evaluation chart |
|  |  |  |  |

**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 creation of a web-based version of the gaming application "Draw It or Lose It" for our customer The Gaming Room is the subject of this software design document. The goal is to make the game available on more platforms than only the one where it is now available as an Android app. To successfully satisfy the client's needs, the suggested solution contains design restrictions, a domain model, and an operating platform evaluation.

## Requirements

*The commercial and technological needs of the customer include memory-efficient solutions for a web-based distributed environment, support for many teams and players, and guaranteeing uniqueness of game and team names.*

## [Design Constraints](#_2et92p0)

Determining the design limitations is essential when creating a gaming application in a distributed online environment. Platform independence, scalability, and security are examples of factors that provide difficulties yet are crucial to take into account. These limitations affect data privacy policies, performance optimization, and development complexity.

## [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 relationships between classes like Entity, Game, Team, and Player are shown in the UML class diagram. It is shown that inheritance and encapsulation are possible, guaranteeing effective software requirements fulfillment. The basic class, the Entity class, facilitates shared characteristics and actions. By using object-oriented concepts to ensure uniqueness, Game represents a game instance with rounds, while Team and Player represent participating teams and players, respectively.

**"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 provides a dependable and intuitive platform for hosting web applications. Reliability and security are guaranteed by its Unix-based design. However, in contrast to other platforms, Mac hosting services might not be as widely available. | Linux offers web-based software applications strong hosting capabilities. Because it's open-source, customization and flexibility are possible. Linux, however, could need more technical know-how for setup and upkeep. | Web-based software applications may be hosted on Windows with broad support and a variety of hosting options. The interface is easy to use, allowing developers to access it. However, in comparison to other platforms, Windows hosting services could be more costly. | Because of their constrained processing and storage capabilities, mobile devices provide a barrier to web-based software applications. Nonetheless, mobile technology has advanced to the point where hosting mobile-friendly, lightweight web apps is now feasible. |
| **Client Side** | It is necessary to consider compatibility with various macOS versions and hardware combinations while developing for Mac clients. Obtaining developer licenses and purchasing Apple gear for testing are potential expenses. Depending on one's level of experience with Apple's programming environment and tools, including Xcode, development times might vary. Proficiency in Objective-C or Swift is important for developing applications for Mac computers. | Compatibility issues with different desktop environments and distributions are part of supporting Linux clients. Because Linux is open-source, costs are often reduced. However, because compatibility testing across distributions is necessary, development time may increase. It is advantageous to have knowledge of Linux programming tools like GCC and Qt. | It is necessary to take into account compatibility with various hardware setups and Windows versions while developing for Windows clients. Purchasing licensing for Windows development tools and doing testing on several Windows versions might incur costs. Depending on how well-versed one is in Microsoft's development tools and environment, including Visual Studio, development times may differ. Proficiency in programming languages like C# and frameworks like.NET is essential for developing software for Windows. | When developing for mobile devices, different platforms like iOS and Android must be taken into account. Obtaining development tool licenses and doing testing on a range of screens and devices may incur costs. Depending on platform-specific needs and upgrades, development times may change. It is required to have knowledge of mobile development languages like Swift, Java, or Kotlin, as well as tools like Android Studio or Xcode. |
| **Development Tools** | Objective-C and Swift are pertinent programming languages and tools for iOS/macOS development, while Xcode is the main integrated development environment (IDE) for Mac development. For cross-platform programming, additional tools like AppCode and Visual Studio Code may be utilized. | C, C++, Python, and Java are useful programming languages and tools for Linux development; IDEs like Eclipse and Visual Studio Code are frequently used. GCC and Make are two more tools that are necessary for Linux software compilation and building. | C#,.NET, and C++ are pertinent programming languages, and Visual Studio is the main integrated development environment (IDE) for Windows development. To facilitate cross-platform development, additional technologies like Xamarin and Visual Studio Code may be utilized. | Depending on the platform, several programming languages and tools are relevant for developing mobile applications. Whereas Android development uses Java or Kotlin and Android Studio, iOS development uses Swift or Objective-C and Xcode. Along with tools like Unity for game creation, cross-platform frameworks like React Native or Flutter may also be taken into consideration. |

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac provides a dependable platform with Unix-based security and reliability, though hosting services may be less widely available. | Linux offers strong hosting capabilities with customization options due to its open-source nature, though it may require more technical expertise. | Windows offers broad support and user-friendly interface for hosting, but services could be costlier compared to other platforms. | Mobile devices have constrained processing and storage, but advancements allow hosting lightweight web apps. |
| **Client Side** | Compatibility across macOS versions and hardware configurations requires consideration, potentially incurring expenses for licenses and hardware. Proficiency in Objective-C or Swift and tools like Xcode is crucial. | Supporting Linux clients involves addressing compatibility across distributions, with costs often reduced due to the open-source nature. Knowledge of Linux programming tools like GCC and Qt is advantageous. | Compatibility with various hardware setups and Windows versions must be ensured, potentially incurring licensing costs for development tools. Proficiency in C#,.NET, and Visual Studio is essential. | Mobile development involves addressing platform-specific needs and testing on diverse devices, requiring licenses for development tools. Proficiency in languages like Swift, Java, or Kotlin and tools like Android Studio or Xcode is necessary. |
| **Development Tools** | Objective-C, Swift, and Xcode are primary for Mac development, with additional tools like AppCode and Visual Studio Code for cross-platform development. | Linux development utilizes C, C++, Python, and Java, with IDEs like Eclipse and Visual Studio Code, along with tools like GCC and Make for compilation. | Windows development relies on C#,.NET, and Visual Studio, with additional technologies like Xamarin for cross-platform development. | Mobile development involves languages like Java or Kotlin for Android and Swift or Objective-C for iOS, with tools like Android Studio or Xcode, along with frameworks like React Native or Flutter for cross-platform development. |

## 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**: Using a cloud-based operating system like Microsoft Azure or Amazon Web Services (AWS) is what I advise. Draw It or Lose It may grow with ease because of these platforms' scalability, versatility, and support for several computer environments, which The Gaming Room provides. Furthermore, cloud platforms offer a multitude of tools for development and deployment across different platforms in addition to strong infrastructure services.
2. **Operating Systems Architectures**: The selected cloud computing platform, like Azure or Amazon, has a distributed architectural paradigm. Its several networked servers, spread across various geographic zones, guarantee fault tolerance and high availability. The architecture consists of parts that are handled centrally via a web-based interface or APIs, including networking infrastructure, storage services, and computing instances.
3. **Storage Management**: A mix of cloud-based storage services, such Azure Blob Storage and Amazon S3 (Simple Storage Service), can be used for storage management. User profiles, video assets, and gaming data may all be stored on these services' scalable, reliable, and highly accessible storage systems. With options for versioning, encryption, and access control rules, data may be arranged into buckets or containers.
4. **Memory Management**: The suggested cloud platform makes use of memory management strategies including resource allocation and virtualization to maximize efficiency and guarantee effective use of system resources. Based on workload needs, instances of the Draw It or Lose It program may be provided with the right amount of memory. Furthermore, cloud platforms include caching services like Azure Redis Cache and Amazon ElastiCache to enhance the responsiveness and performance of applications.
5. **Distributed Systems and Networks**: Network protocols and distributed software architecture can be used to facilitate communication between different platforms. Draw It or Lose It can use message queues or RESTful APIs to enable communication between online clients, mobile devices, and game servers. To provide reliable connectivity, fault tolerance, and graceful degradation in the event of network outages or connectivity problems, the design should take dependencies between components into account.
6. **Security**: To safeguard user data on and across platforms, a variety of security measures should be put in place. Built-in security capabilities like network firewalls, identity and access management (IAM), and encryption both in transit and at rest are available on the suggested cloud platform. To limit access to user accounts and game data, Draw It or Lose It can impose authorization and authentication procedures. Furthermore, sensitive data that is sent across a network or kept in databases can be protected via data encryption techniques. Adherence to industry standards and regulatory requirements is ensured by regular security audits and compliance certifications.