

Draw It, or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 09/19/24 | Ashlyn Saucier | Initial draft of the software design document |
| 2.0 | 10/05/24 | Ashlyn Saucier | Revised Evaluation section (Development Requirements) |
|  |  |  |  |

**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)

* This software design document outlines the development of the "Draw It or Lose It" game application. The app allows users to interactively guess from clues rendered from a library of images. To ensure a fair gaming experience, it is essential that the application manages unique game instances and prevents duplicate names for games and teams. Implementing the Singleton pattern will help maintain a single instance of GameService, centralizing control and preventing conflicts.

## Requirements

* *The game can involve one or more teams.*
* *Each team has multiple players.*
* *Game and team names must be unique to avoid duplicates.*
* *Only one instance of the game can exist in memory at any time, using unique identifiers for each game, team, or player.*

## [Design Constraints](#_2et92p0)

* Scalability: The application must efficiently handle multiple game instances without degrading performance.
* User Accessibility: The interface must remain responsive even with multiple simultaneous users.
* Cross-Platform Compatibility: The game must function seamlessly across various devices and operating systems.
* Data Integrity: Unique naming conventions for games and teams require robust systems for real-time duplicate checks.

## [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 illustrates the relationship between classes in the game application, including Game, Player, and Team. The Entity class serves as an abstraction for common attributes like ID and name, which all entities will inherit. The diagram effectively demonstrates object-oriented principles such as inheritance and encapsulation to meet software requirements.

**"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 is primarily a desktop platform and while it can host servers (e.g., using macOS Server), it is less commonly used for large-scale deployments compared to Linux. It is suitable for development and testing but may face scalability challenges in production environments. | Linux is an excellent choice for server environments, known for its performance, stability, and robustness under high load. Its open-source nature allows customization for specific needs but has a steeper learning curve for new administrators. | Windows Server is widely used and offers extensive support for enterprise-level applications. However, it comes with licensing costs, which can increase overall expenses for The Gaming Room. Windows systems are often user-friendly but may encounter performance limitations with high concurrency. | Mobile devices vary significantly in processing power and resolution. They can be used as clients efficiently but are not suitable for hosting the application server. Network conditions and differences in screen size must be managed to ensure a seamless user experience. |
| **Client Side** | Development on Mac may lead to higher software costs, particularly if using paid development tools like Xcode. However, it benefits from seamless integration with other Apple products and services, enhancing user experience and interface design. | Linux development environments can be cost-effective due to the availability of free and open-source software. However, ensuring compatibility with graphical design tools and gaming libraries may require additional expertise. | Windows offers a vast array of tools and libraries for game development, making it easy for teams familiar with this platform. However, it may present compatibility issues when running the web application on different browsers or across devices. | Mobile development is generally costly and time-consuming due to the need for platform-specific codebases (Android and iOS) and responsive design considerations. Ensuring a consistent user experience across varied devices and resolutions adds to the complexity of development. |
| **Development Tools** | Xcode is essential for software development on Mac, particularly for iOS applications. Utilizing Swift or Objective-C is recommended. However, the Mac ecosystem sometimes limits cross-platform integration with non-Apple products. | A wide range of open-source tools, such as Eclipse and IntelliJ IDEA, support various programming languages (Java, Python), facilitating flexible development options. The learning curve for these tools can be moderate to steep for newcomers. | Visual Studio is a powerful IDE tailored for Windows development, supporting multiple languages. The ecosystem includes a wealth of libraries and frameworks, which can expedite development. However, licensing fees may be a consideration for commercial projects. | Development tools like Android Studio for Android apps and Xcode for iOS are mandatory, and there may be added complexity with development frameworks like React Native or Flutter to ensure cross-platform compatibility. Both ecosystems impose their own constraints and require expertise in mobile development. |

## Recommendations

1. Operating Platform

* I recommend utilizing Linux as the primary operating platform for the expansion of "Draw It or Lose It." Linux is well-regarded within the industry for its reliability, performance, and open-source nature, which allows for extensive customization. Its architecture is particularly optimized for server environments, enabling effective management of concurrent processes, which is essential for a multi-user gaming application that seeks to serve a diverse audience across various platforms.

1. Operating Systems Architectures

* Description of Architecture: The architecture underlying Linux is predominantly characterized as a monolithic kernel. This design choice facilitates direct interaction with hardware, leading to more efficient resource management and process execution.
* Kernel Components: The Linux kernel is responsible for managing core system resources, including CPU scheduling, memory management, and I/O operations, effectively allowing multiple application instances to run concurrently without degrading performance.
* User Space vs. Kernel Space: The distinction between user space (where applications execute) and kernel space (where core operating system functions occur) enhances system stability and security. This architecture reduces the likelihood of user-space applications adversely affecting critical kernel operations.
* Microservices Architecture: Implementing a microservices architecture on Linux can enable different components of the application (e.g., user authentication, game logic, and session management) to operate as independent services. This modularity promotes scalability and ease of maintenance, adapting to the evolving requirements of "Draw It or Lose It."

1. Storage Management

* Identification of Storage System: For effective storage management, I recommend the implementation of PostgreSQL as the relational database management system to complement the Linux operating platform. PostgreSQL is distinguished by its robustness and powerful feature set.
* Data Integrity and Reliability: PostgreSQL’s adherence to ACID properties (Atomicity, Consistency, Isolation, Durability) ensures that user data is handled with the utmost integrity, which is imperative for tracking player interactions and game states accurately.
* Scalability: The database's ability to efficiently manage complex queries and accommodate large datasets makes it especially suitable for the dynamic data needs of the gaming application.
* Customizability: Being open-source, PostgreSQL allows for tailored implementations that can adapt to the specific demands of the application, thus facilitating optimizations to suit performance requirements.

1. Memory Management

* Memory Management Techniques: The Linux operating platform employs effective memory management strategies that align well with the performance needs of "Draw It or Lose It":
* Dynamic Memory Allocation: The use of standard library functions such as `malloc()` and `free()` enables the dynamic allocation of memory resources, accommodating the fluctuating memory needs associated with variable game instances and user connections.
* Paging and Segmentation: These techniques allow Linux to manage memory efficiently by using both logical and physical memory segmentation, optimizing memory allocation and ensuring quick access to active game processes.
* Memory Overcommit Handling: Linux’s configuration options for memory overcommit enable the system to handle varying user loads effectively, ensuring stability and performance are maintained during peak usage periods.

1. Distributed Systems and Networks

* Communication Between Platforms: To facilitate inter-platform communication for "Draw It or Lose It," a distributed systems architecture should be implemented, leveraging RESTful APIs to enable interaction between client applications and the server.
* API Design: The implementation of RESTful APIs will provide clear endpoints for managing game sessions and user data, ensuring that the application can handle interactions seamlessly across various devices.
* Dependency Management: The architecture must account for the interdependencies among system components, implementing robust error-handling mechanisms to address potential network outages or service disruptions. Techniques such as load balancing can mitigate risks associated with user load variability.
* Cloud Integration: Utilizing cloud resources can enhance the application's scalability, allowing for dynamic resource allocation based on user demand, which is vital for a gaming platform expecting variable user activity.

1. Security

* Protecting User Information: Given the critical nature of user data security in online applications, a comprehensive security strategy is paramount. The following recommendations will help safeguard sensitive information:
* Data Encryption: Implementing HTTPS for secure data transmission between clients and servers protects against eavesdropping and man-in-the-middle attacks, thereby ensuring data privacy.
* Authentication Mechanisms: Employing secure authentication protocols such as OAuth 2.0 will safeguard user accounts and prevent unauthorized access. This method provides secure delegated access and mitigates risks related to credential theft.
* Regular Security Audits: Establishing a routine for conducting security audits and vulnerability assessments is essential for identifying and addressing potential threats. This proactive measure will help maintain the integrity and confidentiality of user data.
* Access Control: Utilizing role-based access control (RBAC) will limit access to sensitive information based on user roles and responsibilities, ensuring that only authorized personnel have access to critical data.
* Conclusion:
* By selecting Linux as the operating platform and implementing a structured strategy that encompasses architecture, storage management, memory management, distributed systems, and stringent security protocols, The Gaming Room will be well-positioned to effectively expand "Draw It or Lose It" across multiple computing environments. These recommendations not only enhance scalability and performance but also ensure the security and integrity of user data, fostering a robust gaming experience for users.