

<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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 06/24/2025 | ADIL PATEL |  |

**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 seeks to expand its popular game, Draw It or Lose It, across multiple platforms including web, desktop, and mobile. This software design document proposes a solution that leverages a Linux-based distributed system architecture to ensure scalability, performance, and cross-platform compatibility. Key considerations such as storage, memory management, distributed communication, and security have been addressed to support this multi-platform expansion.

## Requirements

The business requirements include expanding Draw It or Lose It to reach users on web, desktop, and mobile platforms. The technical requirements include supporting real-time multiplayer features, maintaining secure user data, achieving high availability, and supporting integration with cloud-based services.

## [Design Constraints](#_2et92p0)

Developing Draw It or Lose It in a web-based distributed environment requires addressing key design constraints such as ensuring reliable real-time network communication, supporting scalability to handle growing user demand, maintaining cross-platform compatibility across web, desktop, and mobile devices, and implementing strong security measures to protect user data. The application must also be optimized for devices with limited resources and ensure consistent synchronization of game data all platforms.

## [System Architecture View](#_ilbxbyevv6b6)

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 class diagram defines core components such as GameRoom, Player, and GameSession. The GameRoom class manages player membership and session initiation. Each Player instance is associated with game actions, while GameSession coordinates game flow. Object-oriented principles such as encapsulation and inheritance are used to maintain code reusability and modularity.

**"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 servers are generally not used in production environments due to hardware cost and limited server-side tools. While UNIX-based, macOS lacks enterprise-level hosting capabilities compared to Linux. | Linux is the most common OS for web servers due to its stability, open-source nature, and extensive server software compatibility. Ideal for scalable, secure, and distributed environments. | Windows Server supports .NET applications and offers robust GUI tools, but can be more expensive and less customizable than Linux. | Not suitable for server-side hosting due to limited resources and connectivity constraints. |
| **Client Side** | Requires developing macOS-compatible applications or browser-based clients. Development tools are available but may require specific hardware. | Supports browser-based and native apps. Smaller market share for desktop clients; focus should be on web compatibility. | Largest desktop user base. Requires Windows-specific packaging or cross-platform solutions like Electron or Java. | Requires development for both Android (Java/Kotlin) and iOS (Swift), increasing cost and time. |
| **Development Tools** | Uses Xcode for development, supports Swift and Objective-C. IDEs like IntelliJ and VS Code also supported for cross-platform builds. | Offers a wide range of tools including Eclipse, IntelliJ, VS Code. Languages include Python, Java, C++, and web technologies. | Visual Studio is the primary IDE, supports C#, .NET, and C++. Windows Subsystem for Linux (WSL) enhances development flexibility. | Android Studio for Android development; Xcode for iOS. Cross-platform tools like Flutter and React Native are also effective. |

## 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**: To support the scalability and cross-platform deployment of Draw It or Lose It, a Linux-based server platform, specifically Ubuntu Server LTS, is recommended. Linux is widely used for enterprise-level applications and is known for its performance, security, open-source, and support for modern technologies such as containers, cloud services, and automation tools.
2. **Operating Systems Architectures**: Ubuntu Server utilizes a monolithic kernel architecture, where core services like process scheduling, memory management, and device drivers operate in the same memory space. This architecture provides high performance and direct access to hardware resources. Additionally, Linux supports modular kernel extensions, which allows dynamic loading of drivers and features as needed. This flexibility makes it ideal for running a web-based, multi-user game service.
3. **Storage Management**: The best storage solution for this platform is a combination of XFS file systems with LVM for flexible partitioning and storage management. For scalability and fault tolerance, cloud-integrated solutions like Amazon S3, Google Cloud Storage, or NAS can be used to store game assets, player data, and logs. These systems offer automatic backups, encryption at rest, and seamless scaling.
4. **Memory Management**: Linux handles memory efficiently using techniques like paging, virtual memory, and slab allocation. For Draw It or Lose It, which may experience variable loads due to user activity, Linux provides features like OOM and swap space to handle spikes and avoid crashes. Additionally, container-based deployments allow memory limits to be assigned per service, which helps prevent any one process from consuming all available system memory.
5. **Distributed Systems and Networks:** To support communication across multiple platforms, the game should adopt a microservices architecture, using REST APIs or WebSockets for real-time interaction. These services can be containerized and orchestrated via Kubernetes, ensuring load balancing and fault tolerance. Service discovery, heartbeat monitoring, and fallback mechanisms are essential for resilience against network interruptions or component outages. Network layers should use HTTPS to secure all service communication.
6. **Security**: User information should be protected both at rest and in transit. Data stored in databases must be encrypted using AES-256, and user passwords should be hashed with bcrypt or Argon2. For communication, implement TLS/SSL certificates across all endpoints. The application should use OAuth 2.0 or JWT for secure authentication and authorization. On the server side, firewalls, use OSSEC for intrusion detection, and perform regular updates and security patches. Linux’s native user and group permission system further secures the file system from unauthorized access.