

<Draw & Win Web App>

# **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 | 09/16/2024 | Sanefa Amzad | Initial draft of software design document, including executive summary, design constraints, UML description, and platform evaluation. |
| 1.1 | 9/29/2024 | Sanefa Amzad | Revised document to include platform evaluation details, added recommendations for operating platforms, and enhanced analysis of system architecture and security features. |
| 1.2 | 10/13/2024 | Sanefa Amzad | Updated recommendations for Project Three, including operating platform, storage management, memory management, distributed systems, and security. |

**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 their current Android-based game, *Draw It or Lose It*, into a web-based platform that can operate across multiple environments, including Windows, Mac, Linux, and mobile devices. The challenge lies in developing a scalable, efficient, and secure software design that ensures seamless interaction between various systems while maintaining the game's unique business rules - such as managing a single instance of the game in memory and ensuring unique team and player names.

To address these needs, we propose implementing a distributed web application using design patterns like Singleton and Iterator. This approach will ensure only one game instance exists at a time and streamline the management of game, team, and player names. Our solution will also include robust security features to protect user data across platforms, as well as an efficient memory and storage management system to ensure smooth performance. By leveraging these design strategies, The Gaming Room will be able to successfully expand their game to new platforms while maintaining high performance, scalability, and security.

## Requirements

The client, The Gaming Room, requires a web-based version of their Android game *Draw It or Lose It*, capable of supporting multiple platforms. Key business requirements include the ability to host multiple teams and players, enforce unique game and team names, and ensure only one game instance exists in memory. On the technical side, the game must be scalable, support a distributed web environment, and maintain unique identifiers for game, team, and player instances. Additionally, best coding practices and efficient software design patterns should be used for maintainability and performance optimization.

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment presents several design constraints:

1. **Scalability**: The system must support multiple concurrent users and teams, ensuring smooth gameplay without performance degradation. This requires the application to handle concurrency and session management effectively.
2. **Cross-Platform Compatibility**: The game must function seamlessly across different operating systems (Windows, Mac, Linux) and devices (desktop, mobile, tablets). This adds complexity to both the user interface and backend development, requiring responsive design and flexible architecture.
3. **Network Reliability and Latency**: The system must be designed to manage intermittent connectivity issues, particularly with distributed players across various networks, ensuring minimal disruption in gameplay.
4. **Security**: User data, including personal information and gameplay details, must be protected across all platforms. The application must comply with data privacy regulations and ensure secure communication between clients and servers to prevent unauthorized access or data breaches.

These constraints influence the development process by necessitating robust architecture, careful resource management, and attention to detail in ensuring scalability, compatibility, security, and network resilience.

## [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.

A typical architecture for a web-based game application like Draw It or Lose It involves a multi-tier structure. The client-server model includes front-end interfaces for users, a server to handle game logic and data storage, and a cloud-based database for efficient data management. Communication between clients and the server relies on internet protocols, while storage management is handled through a cloud-based solution to ensure data is saved and retrieved efficiently. A logical topology maps out how data flows between users, servers, and storage systems, highlighting key aspects of communication and resource distribution in the application.

## [Domain Model](#_8h2ehzxfam4o)

**"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.**

The UML class diagram for "The Gaming Room" depicts important connections and interactions between entities:

* **Entity Class**: The base class containing common attributes (id, name), promoting reusability and consistency across the system.
* **GameService Class**: Manages a list of games (List<Game>), maintaining identifiers for teams, players, and games using the Singleton pattern (getInstance() method).
* **Game Class**: Contains multiple teams (List<Team>) and manages them using methods like addTeam().
* **Team Class**: Associated with multiple players (List<Player>) and includes methods to manage player additions (addPlayer()).
* **Player Class**: Represents individual players with id and name, showcasing a one-to-many relationship with teams.

This design leverages OOP principles such as inheritance, encapsulation, and the Singleton pattern to manage shared resources.

## [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** | macOS, with its Unix-based foundation, provides robust networking features, stability, and security. It supports hosting web servers like Apache and Nginx, making it suitable for web applications. However, it is not widely used for large-scale server deployments due to high hardware costs and limited compatibility with enterprise software. Better suited for creative industries and smaller environments. | Linux is the preferred choice for server environments due to its open-source nature, flexibility, and high scalability. Distributions like Ubuntu Server, CentOS, and Red Hat Enterprise Linux are optimized for hosting applications. Linux integrates seamlessly with cloud platforms (AWS, Google Cloud, Azure) and supports containerization (e.g., Docker, Kubernetes). | Windows Server integrates with Microsoft products (e.g., Active Directory, SQL Server) and provides a familiar GUI for administration. Hosting through IIS includes built-in security, load balancing, and web support. Integration with Azure enables easy scaling but is more costly due to licensing fees. Best for environments heavily reliant on Microsoft infrastructure. | Mobile devices are typically not used for server-side hosting due to hardware limitations. They act as clients connecting to cloud-based services via REST APIs. Limited server-like functionality can be achieved with tools like Apache Cordova and Node.js, but this is constrained. |
| **Client Side** | macOS provides a high-quality user experience with tools like Xcode and Swift. However, it has a smaller market share compared to Windows, which could limit the audience. Cross-platform frameworks (e.g., Electron, Qt) improve accessibility, though high hardware costs may be a barrier. | Linux offers great flexibility and freedom for client-side development. It supports various desktop environments (e.g., GNOME, KDE) and cross-platform frameworks like Qt and Electron for building applications compatible with macOS, Windows, and Linux. Linux’s open-source nature attracts developers and provides extensive software repositories, but its smaller desktop market share means a narrower reach for client-side applications. Non-technical users may find Linux less intuitive compared to Windows or macOS. | Windows dominates the desktop market, making it the most critical platform for reaching a broad audience. It provides comprehensive development tools like Visual Studio, .NET, and WPF for building client-side applications. The familiar GUI of Windows improves accessibility and user adoption. However, extensive testing is often needed to ensure compatibility across different Windows versions (e.g., Windows 10, Windows 11). Its wide range of hardware options and software compatibility enhances user reach. | Essential for modern applications due to accessibility. Cross-platform frameworks (e.g., React Native, Flutter) ensure consistency across devices. Native development (Xcode for iOS, Android Studio for Android) optimizes performance but requires separate codebases. |
| **Development Tools** | Supports languages like Swift, Objective-C, Java, and C++. Xcode is the primary IDE, with Apple Simulator for device testing. Cross-platform IDEs like IntelliJ IDEA and VS Code are available. | Supports a wide range of languages (Python, C++, Java, JavaScript) and IDEs (VS Code, Eclipse, PyCharm). Open-source nature allows customization. Ideal for server-side development, networking, and web applications. | Supports multiple languages (C#, C++, Python, JavaScript) with Visual Studio, offering tools for .NET, ASP.NET, and UWP. Integrates with databases (SQL Server, MySQL) and cloud services (Azure). | Uses Xcode for iOS and Android Studio for Android. Cross-platform frameworks (React Native, Flutter) streamline development, though native development offers the best performance. |

**Recommendations**

1. **Operating Platform**:

* **Recommendation:**

The Linux operating system is recommended as the primary server platform for "Draw It or Lose It." For hosting a distributed web application, its great scalability, open-source design, and compatibility with numerous development tools make it the perfect choice. Flexibility, scalability, and worldwide accessibility can be obtained by using cloud services such as AWS, Google Cloud Platform, or Azure, which often use servers running Linux.

1. **Operating Systems Architectures**:

* **Details:**

Web browsers and mobile apps will function as clients for the client-server architecture, which manages user management and game logic on the server. Linux is well-suited for effectively managing multiple users and sessions due to its concurrent and multi-threaded processing capabilities. Docker and other containerization solutions make it simple to scale an application across cloud platforms.

1. **Storage Management**

* **Recommendation:**

For storing structured data, such as user accounts, game states, and player interactions, an RDBMS (Relational Database Management System) such as PostgreSQL or MySQL is suggested. Cloud-based storage solutions provide high availability, and Redis caches can improve performance by holding onto data that is accessed often.

1. **Memory Management**:

* **Explanation:**

Linux optimizes memory allocation through the use of memory segmentation and paging. Redis caching can be used to cache frequently accessed data in memory and lessen the strain on the database. In languages like Java, the implementation of garbage collection will also retrieve memory that is not being used, guaranteeing effective memory management.

1. **Distributed Systems and Networks**:

* **Explanation:**

Communication between platforms can be facilitated using RESTful APIs for data exchange or WebSocket connections for real-time interactions. Load balancing, such as that achieved with Nginx, divides network traffic among several servers to keep performance stable even with large user numbers. Monitoring tools for handling network connectivity and any outages should be part of the architecture.

1. **Security**:

* **Explanation:**

Security is essential for protecting user information. Enforcing HTTPS ensures encrypted communication between clients and servers. OAuth 2.0 can be used for secure authentication, and data encryption (e.g., AES) will protect sensitive information in storage. Linux's built-in firewall (iptables) and routine security audits will protect against threats like DDoS attacks and unauthorized access.