

The Gaming Room

# **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 | 05/22/2024 | Ronald Williams | <Brief description of changes in this revision> |

**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 has commissioned Creative Technology Solutions to develop a web-based version of their popular Android game, Draw It or Lose It. The goal is to expand the game's availability across multiple platforms while maintaining the core gameplay mechanics. The game involves teams competing to guess what is being drawn from a library of stock images within a set time limit. This document outlines the software design for the new web-based application, focusing on the essential requirements and constraints.

The critical requirements include support for multiple teams, each with multiple players, and ensuring unique names for games and teams. Furthermore, the application must enforce that only one instance of the game exists in memory at any given time. This will be achieved using unique identifiers for each instance of a game, team, or player.

## Requirements

**Business Requirements**

1. **Platform Expansion:** Expand the game availability from Android-only to web-based platforms, making it accessible via web browsers on desktops and mobile devices.
2. **Core Gameplay Mechanics:** Maintain the core gameplay of Draw It or Lose It, where teams compete to guess stock drawings within a set time limit.
3. **Team Structure:** Enable the creation and management of multiple teams, each consisting of multiple players.
4. **Unique Naming:** Ensure that game and team names are unique to avoid conflicts and provide a seamless user experience.

**Technical Requirements**

1. **Web-Based Application:** Develop the game as a web-based application accessible via standard web browsers.
2. **Singleton Instance:** Ensure only one instance of the game exists in memory at any given time.
3. **Real-Time Validation:** Implement real-time validation to check the uniqueness of game and team names.
4. **Scalability:** Design the application to handle multiple concurrent users efficiently.
5. **Session Management:** Implement session management to handle user sessions securely and efficiently.
6. **Database Integration:** Integrate with a database to store game, team, and player information persistently.
7. **Responsive Design:** Ensure the application is responsive and works seamlessly across various devices and screen sizes.

## [Design Constraints](#_2et92p0)

1. **Web-Based Distributed Environment:**
   * The application must be accessible through web browsers on various platforms, including desktops and mobile devices.
   * **Implications:** The design must consider statelessness, session management, and scalable backend services to handle multiple concurrent users.
2. **Unique Names and Identifiers:**
   * Game and team names must be unique to avoid conflicts and ensure a smooth user experience.
   * **Implications:** The application will need robust validation mechanisms to check the availability of names in real-time.
3. **Singleton Game Instance:**
   * Only one instance of the game can exist in memory at any given time.
   * **Implications:** The singleton pattern will be used to enforce a single instance of the game, ensuring consistency and preventing resource conflicts.

## [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)

1. **GameService Class**
   * **Attributes:**
     + **games: List<Game>**
     + **nextGameId: long**
     + **instance: GameService**
   * **Methods:**
     + **getInstance(): GameService**
     + **addGame(name: String): Game**
     + **getGame(index: int): Game**
     + **getGame(id: long): Game**
     + **getGame(name: String): Game**
     + **getGameCount(): int**
2. **Game Class**
   * **Attributes:**
     + **id: long**
     + **name: String**
   * **Methods:**
     + **getId(): long**
     + **getName(): String**
     + **toString(): String**
3. **ProgramDriver Class**
   * **Methods:**
     + **main(args: String[]): void**
4. **SingletonTester Class**
   * **Methods:**
     + **testSingleton(): void**

**Singleton Pattern: Implemented in the GameService class to ensure that only one instance of the game service exists. This is critical to maintaining the integrity and consistency of the game's state.**

**Iterator Pattern: Used in the GameService methods to traverse the list of games, providing a clean and efficient way to search for games by name or ID.**

**Factory Method Pattern: Utilized in the GameService class to create new game instances, encapsulating the instantiation logic.**

**Composition: The GameService class has a composition relationship with the Game class, indicated by the games attribute, which is a list of Game objects. This denotes that a GameService contains multiple Game instances.**

**Encapsulation: All classes encapsulate their data using private attributes and provide public getter methods to access these attributes, ensuring that the internal state of objects is protected and only accessible through defined methods.**

**"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 can serve as a reliable development and testing environment for web-based applications. It is Unix-based, like Linux, and provides strong security and stability. However, Mac OS is less commonly used for production servers compared to Linux. It’s hardware requirements are typically higher, and the cost of Mac servers can be prohibitive. Mac OS supports a variety of development tools and environments but might not offer as much flexibility and scalability as Linux. | Linux is widely regarded as the best choice for hosting web-based software applications. It is open-source, highly customizable, and supports a wide range of hardware. Linux distributions like Ubuntu, CentOS, and Debian are commonly used in production environments due to their reliability, security, and performance. The cost is minimal as there are no licensing fees, and the vast community support makes it easier to troubleshoot issues. Linux also offers extensive support for various server technologies and programming languages. | Windows Server provides a robust environment for hosting web-based applications, particularly those developed with Microsoft technologies like .NET. It offers excellent support for enterprise applications and integration with other Microsoft services. However, Windows Server can be more expensive due to licensing fees, and it may require more frequent patching and maintenance compared to Linux. It also tends to have a higher resource overhead, which could impact performance. | Mobile devices are generally not suitable for hosting web-based applications due to their limited processing power, storage, and network capabilities. While they are excellent for accessing and interacting with web applications, their use as servers is impractical. Mobile devices are better utilized for client-side interactions where they can leverage responsive web design to provide a seamless user experience. |
| **Client Side** | Developing for Mac clients involves considerations around cost, time, and expertise. The development tools, such as Xcode, are specific to Mac OS, and developers need to be proficient in languages like Swift and Objective-C. Macs are typically more expensive than other PCs, and the user base, while dedicated, is smaller compared to Windows. However, Mac users often expect high-quality, well-designed applications, which may increase development time and cost. | Supporting Linux clients requires knowledge of various distributions and their nuances. Linux users are typically more technical and expect robust, customizable applications. The development cost can be lower due to the use of open-source tools and the absence of licensing fees. However, the fragmentation of Linux distributions can increase the complexity of testing and support. Developers often use tools like Eclipse or Visual Studio Code, and languages such as Python, Java, and C++. | Windows has the largest desktop user base, making it a crucial platform for client-side support. Development for Windows is streamlined with tools like Visual Studio and languages such as C#, .NET, and C++. The cost of development can be higher due to licensing fees and the need for Windows-specific expertise. However, Windows provides a comprehensive development environment and robust support for enterprise applications, which can reduce development time. | Developing for mobile devices involves creating responsive web applications or dedicated mobile apps. The primary considerations are ensuring compatibility across various devices and operating systems (iOS, Android). This requires expertise in mobile development frameworks (e.g., React Native, Flutter) and knowledge of platform-specific languages (Swift for iOS, Kotlin/Java for Android). The development cost and time can be significant due to the need for extensive testing and optimization for different screen sizes and hardware capabilities. |
| **Development Tools** | For Mac, relevant programming languages include Swift and Objective-C. The primary IDE is Xcode, which provides a comprehensive suite of tools for developing, testing, and deploying applications on Mac OS. Other tools like Homebrew for package management and Terminal for command-line operations are also commonly used. | Linux supports a wide range of programming languages such as Python, Java, C++, and JavaScript. Popular IDEs include Eclipse, Visual Studio Code, and IntelliJ IDEA. Tools like Git for version control, Docker for containerization, and Jenkins for continuous integration are essential for development on Linux. | Windows development typically involves languages like C#, .NET, and C++. The primary IDE is Visual Studio, which offers extensive features for developing and debugging applications. Other tools include PowerShell for scripting, Windows Subsystem for Linux (WSL) for cross-platform development, and various SQL servers for database management. | Developing for mobile devices involves using languages like Swift (iOS), Kotlin/Java (Android), and frameworks such as React Native and Flutter for cross-platform development. IDEs include Xcode for iOS, Android Studio for Android, and Visual Studio Code for hybrid development. Tools like Firebase for backend services, Git for version control, and various testing frameworks (e.g., XCTest, Espresso) are also critical.  This evaluation provides a comprehensive overview of each platform's characteristics, advantages, and weaknesses, helping to inform the decision-making process for developing the web-based game application. |

## 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 Recommendation: Linux**

**Linux is the ideal operating platform for expanding Draw It or Lose It to other computing environments due to its versatility, stability, and cost-effectiveness. It supports a wide range of hardware configurations and is highly scalable, making it suitable for both server-side and client-side applications. Linux's open-source nature eliminates licensing fees, which can significantly reduce costs. Furthermore, Linux is known for its robust security features, which are crucial for protecting user information in a web-based environment.**

1. **Operating Systems Architectures** **Description of Chosen Platform Architectures:**

**Linux Architecture:**

**Kernel:** The core of the Linux operating system, managing system resources, and hardware interactions. It handles process scheduling, memory management, and device management.

**System Libraries:** Essential libraries that provide application programs with the functionalities to interact with the kernel. Examples include the GNU C Library (glibc).

**System Utilities:** Basic commands and functionalities for managing files, processes, and system operations (e.g., bash, systemd).

**User Space:** The environment where user applications run. Applications interact with the system through APIs provided by the system libraries.

Linux supports various distributions (e.g., Ubuntu, CentOS, Debian), each tailored for specific use cases, from desktops to servers.

1. **Storage Management**

**Recommendation: Using a Combination of Relational Database and Object Storage**

**Relational Database:**

**Characteristics: Structured data storage with support for complex queries and transactions. Ideal for storing user information, game data, and other structured data.**

**Advantages: ACID (Atomicity, Consistency, Isolation, Durability) compliance ensures data integrity and reliability. SQL provides powerful querying capabilities.**

**Object Storage:**

**Characteristics:** Unstructured data storage for large amounts of binary data (e.g., images, logs). Objects are stored in a flat namespace and accessed via HTTP APIs.

**Advantages:** Scalability, durability, and ease of use. Suitable for storing game assets like stock drawings.

1. **Memory Management**

**Linux Memory Management:**

**Virtual Memory: Linux uses virtual memory to provide each process with its own address space, isolating processes from one another and improving security.**

**Paging and Swapping: Memory pages are used to manage memory efficiently, with the kernel paging out inactive pages to disk (swap space) to free up physical memory.**

**Cache Management: Linux aggressively uses free memory for disk caching to speed up access to frequently used files and reduce I/O operations.**

**Memory Allocation: The kernel employs various algorithms (e.g., buddy system, slab allocator) to allocate and deallocate memory efficiently.**

**These techniques ensure that the Draw It or Lose It application runs smoothly and efficiently, even under heavy load.**

1. **Distributed Systems and Networks Distributed Software and Networking:**

**Microservices Architecture: Decompose the application into smaller, independent services (e.g., user service, game service, team service) that communicate over APIs. This enhances scalability and maintainability.**

**API Gateway: A single entry point for all client requests, routing them to the appropriate microservices. It can also handle cross-cutting concerns like authentication, rate limiting, and logging.**

**Load Balancing: Distribute incoming network traffic across multiple servers to ensure no single server becomes a bottleneck. This improves the application’s availability and reliability.**

**Service Discovery: Use tools like Consul or Kubernetes to manage the dynamic nature of service instances and their locations.**

**Fault Tolerance: Implement retries, circuit breakers, and failover mechanisms to handle partial failures gracefully and ensure high availability.**

**Network Considerations:**

**Connectivity: Ensure reliable and high-speed internet connections for all servers and clients.**

**Outages: Plan for redundancy and disaster recovery strategies to minimize downtime and data loss.**

1. **Security**: **Security Measures:**

**Encryption:** Use HTTPS to encrypt data in transit and ensure secure communication between clients and servers. Encrypt sensitive data at rest using tools like LUKS for disks and database encryption features.

**Authentication and Authorization:** Implement robust authentication mechanisms (e.g., OAuth 2.0, JWT) to verify user identities and control access to resources.

**Firewalls and Intrusion Detection:** Deploy firewalls (e.g., iptables, firewalld) to block unauthorized access. Use intrusion detection systems (e.g., Snort, OSSEC) to monitor and alert on suspicious activities.

**Regular Updates and Patching:** Keep the operating system, libraries, and application dependencies up-to-date to mitigate vulnerabilities.

**Data Backups:** Regularly back up data to prevent loss in case of hardware failure or security breaches.

**Access Controls:** Implement strict access controls and audit logging to track and manage who has access to what resources.