

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 | 10/13/24 | Hayley Foley | Filled out operating platforms, operating system architects, storage management, memory management, distributed system and networks, 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 has requested the development of a web-based version of their game "Draw It or Lose It," where teams compete to guess puzzles based on stock images rendered in real-time. The solution involves designing a system where each game, team, and player has a unique identifier, ensuring smooth and error free participation. To achieve this, implementing the Singleton design pattern to ensure only one game instance runs at any given time. The application will incorporate back-end logic for name uniqueness checks and session handling, along with a front-end interface that provides real-time rendering of images.

## Requirements

The business requirement for this project includes, expanding "Draw It or Lose It" to run on multiple operating systems such as Windows, macOS, Linux, Android, iOS. Ensure a seamless cross-platform gaming experience with efficient memory and storage management. Provide secure storage for game data, including user drawings. Allow scalable and reliable multiplayer support via distributed systems.

The technical requirements for this project include, developing platform-specific client applications that interact with a centralized cloud server, using efficient file system and memory management techniques to optimize performance on all platforms, implementing a distributed system architecture with robust networking to handle real-time data synchronization across devices.

## [Design Constraints](#_2et92p0)

Scalability as the game must support multiple concurrent users while maintaining real-time interactions. This requires efficient server management and cloud-based architecture to handle increasing loads without degrading performance. Implications would include a distributed environment will be necessary, likely hosted on platforms such as AWS or Azure to ensure uptime and scalability.

Singleton Instance, only one instance of the game can exist in memory at any given time. The application must implement a Singleton design pattern to ensure this, affecting how the game is instantiated and managed in memory. Implications would include a careful control over memory and resource allocation to prevent duplicate game instances.

Web-Based Architecture as the game will be web-based, the design will need to consider browser compatibility, session management, and potential latency issues for users with different internet speeds. Implications include a responsive front-end and optimized back-end performance will be critical for delivering a consistent user experience across various platforms.

## [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 ProgramDriver class contains a method +main(), which serves as the entry point for the program. It's responsible for initiating the program by calling relevant methods in other classes.

A <<uses>> relationship with the SingletonTester class is represented by a dashed arrow, indicating that ProgramDriver interacts with SingletonTester but does not have a direct association with it.

The SingletonTester class contains a method +testSingleton() that is used to test the Singleton implementation within the game. This class is responsible for ensuring that the Singleton pattern is correctly implemented in the GameService class. The <<uses>> relationship from ProgramDriver suggests that ProgramDriver calls or utilizes the SingletonTester class to verify the singleton behavior in the system.

The Entity class holds the attributes -id: long, a unique identifier for each object derived from Entity and -name: String, the name of the object. Methods are -Entity(), constructor to initialize the entity, +getId(), +getName(), getter methods to retrieve the id and name of the entity and +toString() which provides a string representation of the entity. The Entity class is a parent class for Game, Team, and Player, as indicated by the inheritance relationships pointing from these classes to Entity.

The GamerService class holds the attributes, -games: List<Game> , holds a collection of Game objects,

-nextGameId, -nextPlayerId, -nextTeamId: long, track the next available ID for games, players, and teams, ensuring that all objects have unique identifiers and -service: GameService, holds a static reference to the Singleton instance of GameService. The methods include, +getInstance(), which implements the Singleton pattern by returning the single instance of GameService.

+addGame(), +getGame(), methods for adding and retrieving Game objects by ID or name.

+getGameCount(), returns the count of games. +getNextPlayerId(), +getNextTeamId(), methods to return the next available player and team IDs. The GameService class is associated with Game through a one-to-many relationship, meaning a single GameService can manage multiple Game objects.

The Game class uses -teams: List<Team> , which holds a list of Team objects associated with the game. Methods include, +Game(), constructor to initialize a game with a unique ID and name. +addTeam(), method for adding teams to the game and +toString(), returns a string representation of the game.

The Game class is associated with the Team class through a one-to-many relationship, meaning a game can have multiple teams.

The Team class holds the attribute -players: List<Player>: which holds a list of Player objects associated with the team. The methods are +Team(), constructor to initialize a team with a unique ID and name. +addPlayer(), method for adding players to the team. +toString(), returns a string representation of the team. The Team class is associated with the Player class through a one-to-many relationship, meaning a team can have multiple players.

The Player class uses the methods +Player(), constructor to initialize a player with a unique ID and name and +toString(), returns a string representation of the player.

**"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 a UNIX-based operating system known for its stability, security, and ease of use. Its integration with development tools which makes it suitable for local development. However, Macs are less commonly used for large-scale web hosting compared to Linux-based servers, and macOS licenses are more expensive. | Linux is widely regarded as the most popular and reliable operating system for hosting web applications due to its stability, flexibility, and open-source nature. It provides excellent performance for servers, supporting a vast array of software and frameworks. However, it has a steeper learning curve, particularly for administrators new to command-line interfaces. Some distributions may also require more manual setup and maintenance compared to proprietary | Windows offers strong compatibility with Microsoft technologies like .NET, making it ideal for web applications built with ASP.NET or C#. It provides a user-friendly interface and robust support for enterprise-level applications. However, Windows hosting comes with higher licensing costs compared to Linux and is less flexible for open-source technologies. | Mobile devices are not typically used for hosting web-based applications due to their limited processing power, storage, and unreliable network connections. However, mobile devices can access web applications through browsers or native apps, offering high portability and convenience for users. While mobile devices are advantageous for running lightweight applications and interacting with cloud-hosted services, they are not designed to handle heavy web hosting tasks, and performance, security, and scalability are significant weaknesses for such use. |
| **Client Side** | Developing software on Mac to support various clients requires expertise in cross-platform frameworks. While Mac offers development environments, the cost of hardware and macOS licenses is high. Development time may increase if teams need to adapt to macOS-specific quirks or support various Apple devices. | Developing on Linux offers significant cost savings due to its open-source nature, with no licensing fees. Expertise in command-line tools, system configuration, and networking is crucial for Linux server management, though many frameworks and tools support cross-platform development, reducing time for multi-client support. Development time is generally efficient due to Linux's compatibility with a wide range of languages and environments, but the learning curve for developers unfamiliar with Linux can increase training costs. | Developing web applications on Windows can be more expensive due to licensing fees for Windows Server and development tools. Time spent developing on Windows may vary depending on whether the application leverages Microsoft's proprietary tools, such as Visual Studio, which can streamline development. | Developing for mobile devices involves considering multiple platforms, primarily Android and iOS, which requires expertise in both ecosystems. Cross-platform frameworks like Flutter, React Native, and Xamarin can reduce development time and costs by supporting both platforms from a single codebase |
| **Development Tools** | For developing and deploying software on Mac, key programming languages include Swift for native macOS/iOS apps, and Python, Ruby, or JavaScript for web applications. Popular IDEs and tools include Xcode for macOS/iOS development, Visual Studio Code for general development, and Homebrew for package management. | Popular programming languages for developing and deploying on Linux include Python, Java, Ruby, PHP, JavaScript. Linux also supports containerization technologies such as Docker and orchestration tools like Kubernetes | Key languages for Windows-based development include C#, VB.NET, JavaScript (Node.js), and PHP. Microsoft’s Visual Studio is the primary IDE for Windows development, offering extensive support for .NET applications. | For mobile development, popular programming languages include Swift for iOS, Kotlin/Java for Android, and JavaScript for cross-platform apps with frameworks like React Native. Tools like Xcode for iOS, Android Studio for Android, and Visual Studio Code for React Native are essential for 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**: For Draw It or Lose It, I recommend using a Linux-based operating platform. Linux offers flexibility, scalability, and extensive support for web-based applications, making it ideal for expanding to different computing environments such as desktop, mobile, and cloud.
2. **Operating Systems Architectures**: The recommended Linux architecture uses a modular structure that allows for customization depending on the environment. It operates on the POSIX-compliant UNIX-like structure, ensuring compatibility with other UNIX-based systems like macOS. The architecture supports multi-user and multitasking environments, enabling the game to run efficiently across distributed platforms.
3. **Storage Management**: For storage management, using a cloud-based solution like Amazon S3 or Google Cloud Storage with Linux will provide scalability and high availability.
4. **Memory Management**: Linux implements efficient memory management techniques, such as paging and demand loading, which allows for optimal memory usage by loading data into RAM only when needed. Virtualmemory is also used, enabling the operating system to use disk space for temporary storage when the RAM is insufficient. For the Draw It or Lose It software, Linux will handle memory allocation for the game’s processes dynamically, ensuring smooth operation even when multiple instances of the game are running simultaneously.
5. **Distributed Systems and Networks**: The Draw It or Lose It game can leverage microservices architecture across distributed systems for seamless communication between platforms. This can be accomplished using RESTful APIs or WebSocket connections to handle real-time communication between clients and the server. To ensure connectivity across distributed environments, load balancing and failover mechanisms can be implemented to maintain service during network outages.
6. **Security**: Linux-based systems offer security features, such as firewalls, SELinux, and AppArmor, which can protect the game’s servers from unauthorized access. To protect user data, encryptionprotocols should be employed to secure communication between platforms. User authentication and access control can be managed using OAuth or JWT to ensure secure login.