

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  2.0 | 7/21/2024  **8/4/2024** | Matthew Uncapher | Initial draft of the design document  Updated Evaluation and Recommendations. |

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 develop a web-based version of their game, Draw It or Lose It, which is currently available only as an Android app. The new game application must be capable of serving multiple platforms. The game involves teams competing to guess what is being drawn, and the application renders images from a large library of stock drawings as clues. The software design will ensure that the game operates smoothly in a web-based distributed environment, leveraging the singleton pattern to manage game instances and ensuring unique identifiers for games, teams, and players.

## Requirements

The client’s business and technical requirements include:

* The game must support multiple teams, each with multiple players.
* Game and team names must be unique to avoid conflicts.
* Only one instance of the game should exist in memory at any time.
* The game should be accessible on various platforms, including web and mobile devices.
* The system should manage unique identifiers for each game, team, and player.
* The application must ensure the security and privacy of user data.

## [Design Constraints](#_2et92p0)

The design constraints for developing the game application in a web-based distributed environment include:

* Scalability: The application must handle multiple teams and players efficiently.
* Uniqueness: Game and team names must be unique to avoid conflicts.
* Singleton Pattern: Ensure only one instance of the game exists in memory at any given time.
* Platform Compatibility: The game must be accessible across different platforms, including web and mobile.
* Performance: The application must perform efficiently under high user load.
* Security: User data must be protected 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 UML class diagram provided shows the structure of the game application. The classes include:

* Entity: A base class holding common attributes and behaviors (id and name).
* GameService: Manages game instances and ensures a single instance of the game using the singleton pattern.
* Game: Represents a game with multiple teams.
* Team: Represents a team with multiple players.
* Player: Represents a player in a team.
* ProgramDriver: The main class to start the application.
* SingletonTester: A class to test the singleton pattern implementation.

The relationships among these classes demonstrate object-oriented programming principles such as inheritance (the Entity class), encapsulation (private attributes with public getters and setters), and the singleton pattern (GameService class).

**"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.

| Development Requirements | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| Server Side | |  | | --- | | Mac offers a robust and secure environment for hosting web-based applications, but can be more expensive and less commonly used in enterprise environments compared to Linux. | |  | | |  | | --- | | Linux is highly popular for server environments due to its stability, security, and cost-effectiveness. It supports a wide range of web technologies. | |  | | |  | | --- | | Windows servers provide good support for .NET applications and integration with other Microsoft services but can be more expensive. | |  | | |  | | --- | | Mobile devices typically act as clients rather than servers in a distributed environment. | |  | |
|  | **Licensing Costs:** Higher licensing costs due to proprietary software. | |  | | --- | | **Licensing Costs:** Open-source, generally free, making it cost-effective. |  |  | | --- | |  | | |  | | --- | | **Licensing Costs:** High due to the cost of Windows Server and associated software. |  |  | | --- | |  | | |  | | --- | | **Licensing Costs:** N/A for server-side, as mobile devices are clients. |  |  | | --- | |  | |
| **Client Side** | |  | | --- | | Mac clients are known for their reliability and high performance in graphics and media applications. Development can be more costly due to hardware and software costs. | |  | | |  | | --- | | Linux clients are less common but can be highly customizable. Development can be complex due to various distributions. | |  | | |  | | --- | | Windows clients are ubiquitous, providing broad compatibility with a range of software. Development is generally more straightforward. | |  | | |  | | --- | | Mobile devices require careful consideration of performance and compatibility. Development can be fragmented due to different OS versions and device capabilities. | |  | |
|  | |  | | --- | | **Development Requirements:** Higher cost for Mac hardware and development tools. Expertise in macOS and iOS development is necessary. |  |  | | --- | |  | | |  | | --- | | **Development Requirements:** Knowledge of various Linux distributions is needed. Customization can be complex but allows for a tailored experience. |  |  | | --- | |  | | |  | | --- | | **Development Requirements:** Widely understood and supported, making development more straightforward. Visual Studio provides comprehensive support. |  |  | | --- | |  | | |  | | --- | | **Development Requirements:** Expertise in mobile app development and responsive web design. Cross-platform tools can streamline development. |  |  | | --- | |  | |
| **Development Tools** | |  | | --- | | macOS development typically uses Xcode for iOS applications. For web applications, tools like IntelliJ IDEA or Visual Studio Code can be used. | |  | | |  | | --- | | Linux development can utilize a variety of tools like Eclipse, IntelliJ IDEA, and command-line utilities, providing flexibility and control. | |  | | |  | | --- | | Windows development often uses Visual Studio, which offers a comprehensive suite of tools for various types of applications. | |  | | Mobile development can use tools like Android Studio for Android and Xcode for iOS, or cross-platform tools like Flutter and React Native. |
|  | |  | | --- | | **Impact on Development Team:** Higher costs for licenses and hardware. May require a specialized team for macOS/iOS. |  |  | | --- | |  | | |  | | --- | | **Impact on Development Team:** Flexible but may require diverse skill sets. Low cost for tools. |  |  | | --- | |  | | |  | | --- | | **Impact on Development Team:** Generally easier to find skilled developers. Higher licensing costs for tools like Visual Studio. |  |  | | --- | |  | | **Impact on Development Team:** May require multiple teams or developers skilled in cross-platform tools to ensure compatibility across devices. |

### **Additional Notes:**

* **Linux:** Best suited for server-side deployment due to its stability, security, and low cost.
* **Mac:** Ideal for client-side development where high performance and graphics are required, but comes with higher costs.
* **Windows:** Offers a balance for both server-side and client-side with good support for various applications but at a higher cost.
* **Mobile Devices:** Development requires careful planning for performance and compatibility, leveraging cross-platform tools can be beneficial.

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

#### **Operating Platform:**

* **Linux** is recommended for server-side development due to its stability, security, and cost-effectiveness. It is highly suitable for hosting the web-based game application as it supports a wide range of web technologies and offers robust performance.
* **Windows** is recommended for client-side development because of its ubiquity and ease of development. Its broad compatibility with various software and ease of use for developers makes it a practical choice for ensuring the application can reach a wide audience.

#### **Operating Systems Architectures:**

* **Server-Side Architecture:** Linux server environments should be used for hosting the web-based game application. This choice leverages Linux's strengths in stability, security, and cost-effectiveness, making it an ideal platform for handling server-side operations efficiently.
* **Client-Side Architecture:** Windows should be used for client development to ensure broad compatibility and ease of development. This will facilitate a smooth user experience on desktop platforms, taking advantage of Windows' extensive user base and developer tools.

#### **Storage Management:**

* **Relational Database Management System (RDBMS):** Use MySQL or PostgreSQL for efficient storage and retrieval of game data. These RDBMS options provide reliable performance, robust data integrity, and scalability, which are essential for managing the game’s data needs effectively.

#### **Memory Management:**

* **Linux Memory Management:** The Linux operating system efficiently manages memory through paging and swapping. Additionally, the Java Virtual Machine (JVM) used for running the application provides garbage collection to manage memory, ensuring optimal performance and resource utilization.

#### **Distributed Systems and Networks:**

* **RESTful APIs:** Implement RESTful APIs to allow communication between different platforms. This approach enables the application to be platform-agnostic, ensuring that various clients (web, mobile) can interact with the server seamlessly.
* **Load Balancing and Redundancy:** Use load balancing and redundancy to ensure high availability and reliability. This will distribute the workload evenly across multiple servers and provide failover capabilities, ensuring the game remains accessible even during peak usage times or server failures.

#### **Security:**

* **Data Encryption:** Implement encryption (e.g., SSL/TLS) to protect data in transit. This will secure the communication between the client and server, preventing unauthorized access and data breaches.
* **Authentication and Authorization:** Use robust authentication and authorization mechanisms to protect user data. Techniques such as OAuth for authorization and JWT (JSON Web Tokens) for secure token-based authentication can be employed.
* **Regular Updates and Patching:** Regularly update and patch systems to protect against vulnerabilities. This proactive approach will help safeguard the application from potential security threats and exploits.