

Draw It or Lose It

# **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 | 05/26/2024 | Jesse James Aranda | Initial draft of the software design document. |
| 1.1 | 06/05/2024 | Jesse James Aranda | Software design document update. |
| 1.2 | 06/21/2024 | Jesse James Aranda | Recommendations update. |
| 1.3 | 06/27/2024 | Jesse James Aranda | Final draft. |

**[Executive Summary](#_sbfa50wo7nsh)**

The Gaming Room aims to develop a web-based version of their popular game, Draw It or Lose It, currently available only on Android. The goal is to create a multi-platform game that allows multiple teams and players, with unique names to prevent duplicates. The game will follow a strict rule where only one instance of the game exists in memory at any time. To achieve this, a robust software design adhering to the client’s requirements and leveraging modern design patterns is essential. This document outlines the software design solution, addressing the critical aspects needed to proceed with the project effectively.

## Requirements

* **Business Requirements**:
  + Develop a web-based version of Draw It or Lose It.
  + Support multiple teams and players.
  + Ensure game and team names are unique.
  + Restrict to one instance of the game in memory.
* **Technical Requirements**:
  + Implement a singleton pattern for the game instance.
  + Use unique identifiers for games, teams, and players.
  + Ensure cross-platform compatibility (web, mobile, desktop).

## [Design Constraints](#_2et92p0)

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

1. **Scalability**: The application must handle multiple concurrent users efficiently.
2. **Performance**: The game must render images and manage game states seamlessly across platforms.
3. **Security**: User data and game states must be secure during transmission and storage.
4. **Compatibility**: The application must run consistently across different operating systems and devices.
5. **Maintainability**: The codebase should be modular and easy to update or extend.

These constraints impact the application development process, necessitating careful selection of technologies and architectural patterns to ensure the system meets performance, security, and maintainability standards.

## [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 provided UML class diagram illustrates the structure of the game application, showcasing relationships and dependencies between its main classes. At the core of the diagram is the ‘**Entity’** class, serving as a base class with common attributes like ‘**id’** and ‘**name’**. This inheritance relationship ensures that subclasses ‘**Game’**, ‘**Team’**, and ‘**Player’** inherit these attributes, promoting code reuse and consistency throughout the application. The ‘**GameService’** class manages the game instances and is designed as a singleton, ensuring only one instance exists at any given time. This implementation adheres to the singleton pattern, a fundamental object-oriented programming principle, which ensures centralized control over game management and prevents duplication of game instances. The ‘**Game’** class represents a game entity that can contain multiple ‘**Team’** objects, showcasing a composition relationship. Similarly, a ‘**Team’** can contain multiple ‘**Player’** objects, demonstrating another composition relationship. These relationships facilitate the organization and management of game entities, teams, and players within the application. Additionally, encapsulation is evident in each class, as they encapsulate their data and expose only essential methods to interact with that data. This promotes modularity and enhances the maintainability and scalability of the software, fulfilling the software requirements efficiently.

**"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)

**Server Side:**

* **Mac:** Offers stable and secure servers but is less common and more expensive.
* **Linux:** Highly reliable, secure, and cost-effective, making it ideal for hosting web applications.
* **Windows:** User-friendly, integrates well with Microsoft products but can be costly.
* **Mobile Devices:** Impractical for hosting; should act as clients.

**Client Side:**

* **Mac:** Requires macOS-specific tools and targets a smaller market segment.
* **Linux:** Needs knowledge of different distributions and has a smaller user base.
* **Windows:** Benefits from a large user base and extensive support for various tools and languages.
* **Mobile Devices:** Requires development for both iOS and Android platforms with distinct environments.

**Development Tools:**

* **Mac:** Uses Swift, Objective-C, Xcode, and macOS SDKs.
* **Linux:** Utilizes C, C++, Java, Python, and various IDEs.
* **Windows:** Supported by .NET, C#, Visual Studio, and other tools.
* **Mobile Devices:** Java/Kotlin for Android and Swift for iOS with respective IDEs (Android Studio, Xcode).

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are stable and secure but are less common and more expensive compared to other options. | Linux servers are highly reliable, secure, and cost-effective, making them ideal for hosting web applications. | Windows servers are user-friendly and integrate well with other Microsoft products but can be costly. | Hosting on mobile devices is impractical due to resource limitations; mobile devices should act as clients rather than servers. |
| **Client Side** | Developing for Mac requires macOS-specific tools like Xcode. Mac users are a smaller market segment. | Developing for Linux requires knowledge of different distributions. Linux has a smaller user base but is favored by developers. | Windows development benefits from a large user base and extensive support for various tools and languages. | Developing for mobile devices requires consideration of both iOS and Android platforms, each with distinct development environments. |
| **Development Tools** | Swift, Objective-C, Xcode, and macOS SDKs are essential tools for Mac development. | C, C++, Java, Python, and a variety of IDEs (like Eclipse and IntelliJ IDEA) are used for Linux development. | .NET, C#, Visual Studio, and a wide range of other tools support Windows development. | Java/Kotlin for Android and Swift for iOS, along with Android Studio and Xcode for their respective platforms. |

## Recommendations

Linux is recommended for its robustness, scalability, and cost-effectiveness, making it ideal for hosting web applications. It features a modular architecture, advanced memory management techniques, and strong security measures. Cloud-based storage solutions like Amazon S3 or Google Cloud Storage offer scalability and reliability. To enable seamless cross-platform communication, use RESTful APIs and WebSockets. Security is ensured through encryption (TLS), secure authentication (OAuth), and regular security audits.

1. **Operating Platform**: Linux offers a robust, scalable, and cost-effective platform for hosting web applications. Its reliability and security make it a suitable choice for the server-side environment. Linux's open-source nature and extensive community support ensure continuous improvements and updates, making it ideal for The Gaming Room's needs.
2. **Operating Systems Architectures:** Linux architecture is modular and supports various distributions tailored to specific needs. It efficiently manages resources and provides strong security features. Key characteristics include:

* **Kernel:**  Manages system resources and hardware communication.
* **File System:** Uses EXT4, Btrfs, and other file systems for efficient data storage.
* **Package Management:** Facilitates software installation and updates through package managers like APT and YUM.
* **Security:**  Implements strong user permission models, SELinux/AppArmor for access control, and iptables for firewall management.

1. **Storage Management**: For storage management, a cloud-based solution such as Amazon S3 or Google Cloud Storage is recommended. These platforms provide:

* **Scalability:** Seamless scaling to accommodate growing data needs.
* **Reliability:** High availability and redundancy to ensure data integrity.
* **Accessibility:** Easy access from different platforms and devices.
* **Security:** Built-in encryption and access control mechanisms to protect data.

1. **Memory Management**: Linux employs advanced memory management techniques to optimize the performance of applications like Draw It or Lose It. These techniques include:

* **Virtual Memory:** Allows applications to use more memory than physically available by swapping data to disk.
* **Paging:** Divides memory into pages to manage data more efficiently.
* **Swap Space:** Extends physical memory onto the disk to prevent out-of-memory errors.
* **Caching:** Stores frequently accessed data in memory to speed up access times.

1. **Distributed Systems and Networks**: To enable seamless communication between various platforms, the following technologies should be implemented:

* **RESTful APIs:** Allow different components of the system to communicate over HTTP, providing interoperability between platforms.
* **WebSockets:** Enable real-time, bidirectional communication between the server and clients, ensuring timely updates and interactions.
* **Load Balancers:** Distribute incoming traffic across multiple servers to ensure high availability and reliability.
* **CDNs (Content Delivery Networks):** Enhance content delivery speed and reliability by distributing data across multiple geographic locations.

1. **Security**: Protecting user information and ensuring secure interactions between various platforms is crucial. The following measures are recommended:

* **Encryption:** Use TLS for secure data transmission over networks.
* **Authentication:** Implement OAuth for secure user authentication and authorization.
* **Regular Security Audits:** Conduct regular audits to identify and address vulnerabilities.
* **Best Practices:** Adhere to industry standards for security, such as OWASP guidelines, to safeguard the application against common threats.