

# 

# Draw It or Lose It 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 | 11/13/2024 | Davit Mumladze | First draft for the software design document |
| 1.1 | 11/29/2024 | Davit Mumladze | Updated document for requirements, and recommendations.. |
| 1.2 | 12/14/2024 | Davit Mumladze | Further refined the 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 is looking to expand their Android game, Draw It or Lose It, into a web-based application that supports multi-platform usage, including Windows, Mac, Linux, and mobile devices. The primary objective is to develop a scalable, multi-platform game that retains the features of the original while expanding accessibility.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

* **Concurrency:**
  + Multiple users will access the game simultaneously, requiring the application to handle concurrent operations safely.
  + Implement thread-safe mechanisms to manage shared resources, such as synchronized methods or locks, to prevent race conditions and data inconsistencies.
* **Scalability:**
  + The application must scale efficiently with an increasing number of users and teams.
  + Use scalable server technologies and design patterns that allow horizontal scaling, such as load balancing and distributed databases.
* **Platform Compatibility:**
  + The game must function seamlessly across Windows, Mac, Linux, and mobile devices.
  + Adopt cross-platform development frameworks and web standards to ensure consistent behavior across all platforms.

**Singleton Pattern Constraint**

* + Only one instance of the GameService class should exist to manage the game's global state.
  + Implement the Singleton pattern carefully to prevent multiple instances, ensuring thread safety in a concurrent environment.
  + Consider potential issues with serialization and cloning that could create additional instances.

**Unique Identifiers and Names**

* + Game, team, and player names and IDs must be unique to prevent conflicts.
  + Implement mechanisms to check for existing names before adding new entities, which may involve database queries or in-memory data structure traversal.
  + Optimize these checks to minimize performance overhead, possibly through indexing or hashing techniques.

**Iterator Pattern Constraint**

* + Efficient traversal of collections is necessary for managing entities and checking uniqueness.
  + Use the Iterator pattern to traverse collections of games, teams, and players.
  + This approach improves code maintainability and allows for flexible iteration over different data structures.

**Resource Limitations**

* **Memory and Processing:**
  + The application must perform well on devices with varying capabilities.
  + Optimize code for efficiency, manage memory usage carefully, and minimize resource-intensive operations.

**Security Considerations**

* **Data Integrity:**
  + Ensure that game data remains accurate and consistent despite concurrent access.
  + Implement transaction management and validation checks to maintain data integrity.
* **User Authentication:**
  + Protect user accounts and personal information.
  + Incorporate secure authentication mechanisms, such as HTTPS, encryption, and secure session management.

## [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 for Draw It or Lose It illustrates the relationships between the classes in the application

**"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** servers offer robust security and stability due to their Unix-based architecture. They are excellent for development and testing environments, especially for applications targeting Apple ecosystems. However, Macs are less commonly used as production servers. This can lead to limited support for certain server software and higher hardware and maintenance costs compared to other platforms. |  |  | | --- | |  | | **Linux** is the predominant choice for server environments. It is open-source, cost-effective, and highly customizable, making it ideal for hosting web-based applications. Linux offers excellent performance, scalability, and a wide range of server software options. The main weakness is the requirement for expertise in Linux system administration. Additionally, variations across different distributions may cause inconsistencies that need to be managed | **Windows** servers provide a user-friendly interface and good support for .NET applications and other Windows-specific technologies. However, they come with higher licensing costs and are generally considered less stable and secure compared to Unix-based systems. Windows is not as widely used for hosting web applications, which may result in less community support and fewer resources for troubleshooting | Mobile devices are typically not used as servers due to their limited resources, such as CPU power, memory, and storage capacity. Hosting a web-based application on a mobile device is impractical and inefficient. Therefore, mobile devices are unsuitable for server-side deployment of the game application and are better utilized as client-side devices. |
| **Client Side** | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  | | --- | | Development for Mac requires ensuring compatibility with macOS browsers and applications. Mac hardware is needed for testing, increasing costs. Tools like Xcode are free but tied to macOS. |  |  | | --- | |  | |   . |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Supporting Linux clients requires ensuring compatibility across various Linux browsers and environments. Costs are lower with open-source tools, but testing can be complex. |  |  | | --- | |  | | | |  | | --- | |  | | **Windows** has the largest user base, making compatibility essential. Widely available tools like Visual Studio reduce time but may have licensing costs for professional versions. | |  |  | | --- | |  | | Mobile devices need responsive design to handle varying screen sizes and touch interfaces. iOS development requires a $99/year subscription; Android development is free. Expertise in responsive design and frameworks is essential. |
| **Development Tools** | |  | | --- | | For **Mac**, relevant programming languages include Java (for cross-platform compatibility) and Swift (for native macOS or iOS applications). Commonly used IDEs are Eclipse and IntelliJ IDEA for Java development, and Xcode for Swift and Objective-C. Additional tools include Homebrew for package management and Terminal for command-line operations. |  |  | | --- | |  | | |  | | --- | | On **Linux**, developers utilize programming languages such as Java, Python, and C++. IDEs like Eclipse, IntelliJ IDEA, and NetBeans are popular choices. Essential tools include GCC for compiling, Make for build automation, and package managers like apt or yum. Linux also offers powerful command-line utilities and scripting capabilities for efficient development and deployment. |  |  | | --- | |  | | |  | | --- | | For **Windows**, programming languages like Java, C#, and other .NET languages are prevalent. IDEs such as Eclipse and IntelliJ IDEA support Java development, while Visual Studio is the primary IDE for C# and .NET applications. Development tools include Visual Studio tools, PowerShell for scripting, and an array of GUI-based utilities to facilitate development and deployment. |  |  | | --- | |  | | **Mobile Devices** development involves programming languages like Java or Kotlin for Android, Swift or Objective-C for iOS, and JavaScript for cross-platform web applications. IDEs include Android Studio for Android development, Xcode for iOS development, and Visual Studio Code for web applications. Essential tools are device emulators, SDKs for respective platforms, and responsive design frameworks like Bootstrap or React Native to ensure compatibility across various devices and screen sizes. |

## 

## Development tools

**Mac**

**Tools**: Xcode (for native applications), IntelliJ IDEA, Eclipse.

**Languages**: Java for cross-platform development, Swift for macOS/iOS applications.

**Licensing Costs**: Development tools like Xcode are free but require macOS hardware, which increases costs.

**Linux**

**Tools**: Eclipse, IntelliJ IDEA, NetBeans, GCC for compiling, Make for build automation.

**Languages**: Java, Python, C++ for development.

**Licensing Costs**: Free and open-source tools keep costs low, but operational costs for enterprise-level solutions may increase.

**Windows**

**Tools**: Visual Studio (Professional version licensing costs may apply), Eclipse, IntelliJ IDEA.

**Languages**: Java for cross-platform development, C#, and other .NET languages for Windows-specific technologies.

**Licensing Costs**: High due to Windows Server licenses and potentially Visual Studio professional tools.

**Mobile Devices**

**Tools**: Android Studio for Android, Xcode for iOS, React Native for cross-platform apps.

**Languages**: Java or Kotlin (Android), Swift or Objective-C (iOS), JavaScript for cross-platform web apps.

**Licensing Costs**: Android development is free. iOS development requires a $99/year Apple Developer Program subscription.

## 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**

Use a **Linux-based server environment** (e.g., Ubuntu Server or CentOS).

* **Justification**:
  + **Cost-Effective**: Linux is open-source and free, reducing costs significantly compared to proprietary solutions like Windows Server.
  + **Scalability**: Easily handles increasing loads, a critical need for supporting a growing user base.
  + **Performance**: Known for its robustness, Linux provides high performance with low resource overhead.
  + **Security**: Strong security features like SELinux and frequent updates make it a reliable choice for hosting web applications.
  + **Cross-Platform Accessibility**: A web application hosted on Linux ensures compatibility across all devices and platforms with modern browsers.

1. **2. Operating Systems Architectures**

* **Server-Side Architecture**:
  + **Linux Distribution**: Ubuntu Server or CentOS.
  + **Key Features**:
    - **Kernel**: Monolithic kernel for efficient hardware interaction.
    - **Process Management**: Support for multithreading and multitasking ensures smooth handling of concurrent users.
    - **File Systems**: Use ext4 or XFS for reliable and efficient data storage.
    - **Security Modules**: SELinux or AppArmor for access control and enhanced security.
* **Client-Side Architecture**:
  + **Browser-Based**: Use frameworks like React or Angular for the user interface.
  + **Progressive Web App (PWA)**: Enhance mobile device user experience by integrating offline capabilities and app-like functionality.
  + **Responsive Design**: Ensure the interface adapts seamlessly to various screen sizes and orientations.

1. **3. Storage Management**

* Implement a **Relational Database Management System (RDBMS)** such as **MySQL** or **PostgreSQL**.
* **Justification**:
  + **Data Integrity**: ACID-compliance ensures consistent and reliable data management.
  + **Scalability**: Both databases support horizontal scaling and clustering.
  + **Performance Optimization**: Features like indexing for fast search and replication for high availability.
* **Implementation**:
  + Enforce unique constraints at the database level for game, team, and player identifiers.
  + Use connection pooling to efficiently manage simultaneous database queries.

1. **4. Memory Management**

* **Server-Side Memory Management**:
  + Leverage **Java’s Garbage Collection** to automatically manage memory, reducing risks of memory leaks.
  + Optimize code to minimize memory consumption by:
    - Using primitives over objects where applicable.
    - Managing object references effectively.
  + Implement caching with **Redis** or **Memcached** to store frequently accessed data, reducing database queries.
* **Client-Side Memory Optimization**:
  + Optimize front-end resources like images and scripts.
  + Use browser caching to enhance response times for repeated user visits.

1. **5. Distributed Systems and Networks**

* **Achieving Interconnectivity**:
  + Use **RESTful APIs** for seamless communication between client and server.
  + Employ **JSON** for lightweight, language-independent data exchange.
* **Network Infrastructure**:
  + **Load Balancing**: Distribute traffic across servers to ensure high availability and fault tolerance.
  + **Content Delivery Networks (CDNs)**: Speed up content delivery for users in different geographic locations.
* **Dependencies and Connectivity**:
  + Implement redundancy to mitigate risks from server outages.
  + Use monitoring tools to detect and resolve connectivity issues proactively.
  + Design systems with graceful degradation to ensure partial functionality during failures.

1. **6. Security**

* **Protecting Data in Transit**:
  + Enforce **SSL/TLS encryption** for all client-server communications.
  + Use HTTPS exclusively to protect user data.
* **Authentication and Authorization**:
  + Implement **OAuth 2.0** or **JWT tokens** for secure user authentication and session management.
  + Use **Role-Based Access Control (RBAC)** to restrict access based on user roles.
* **Protecting Data at Rest**:
  + Encrypt sensitive data using AES standards.
  + Hash passwords with algorithms like bcrypt or Argon2.
* **Platform-Level Security**:
  + Use Linux-specific security features like firewalls (iptables, ufw) and intrusion detection systems (Fail2Ban).
  + Regularly update and patch systems to address vulnerabilities.
* **Application Security**:
  + Sanitize all user inputs to prevent SQL injection and cross-site scripting (XSS).
  + Conduct regular security audits and penetration testing.
* **Compliance**:
  + Adhere to data protection regulations such as GDPR or CCPA.
  + Develop and display clear privacy policies for users.