

Draw It or Lose It Web Application

# **CS 230 Project Software Design**

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 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

[**Recommendations 10**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 6/9/24 | Omer Cengiz | Initial creation of the software design document. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to expand their game, Draw It or Lose It, to a web-based platform that supports multiple computing environments. The game consists of teams competing to guess drawings rendered by the application. This document outlines the software design to achieve this goal by detailing the system architecture, design constraints, and the implementation of key software design patterns. The primary objective is to create a scalable, efficient, and secure application that can handle multiple teams and players, ensuring unique identifiers and maintaining the game's integrity.

## Requirements

The game application must meet the following requirements:

* Support multiple teams, each consisting of multiple players.
* Ensure that game and team names are unique.
* Implement a singleton pattern to maintain a single instance of the game service.
* Use the iterator pattern to manage collections of games and teams.

## [Design Constraints](#_2et92p0)

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

* Scalability: The system must efficiently handle a large number of concurrent users and teams.
* Resource Management: Efficient memory and processing resource management is critical to ensure smooth operation.
* Security: Secure handling of user data across distributed systems is paramount.

## Performance: The application must render drawings and process user inputs quickly to maintain a smooth gaming experience.

## [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 represents the relationships between different classes in the game application. Key classes include:

* Entity: A base class with common attributes (id and name) and methods.
* GameService: Manages the list of games and generates unique IDs. Implements the Singleton pattern.
* Game: Contains a list of teams and methods to add teams.
* Team: Contains a list of players and methods to add players.
* Player: Represents a player with an ID and name.
* ProgramDriver: Contains the main() method to run the application.
* SingletonTester: Contains the testSingleton() method to test the Singleton pattern.

Object-Oriented Programming Principles:

* Inheritance: The Game, Team, and Player classes inherit from the Entity class to reuse common attributes and methods.
* Encapsulation: Each class hides its internal implementation details and exposes only necessary methods.
* Singleton Pattern: Ensures only one instance of GameService exists.
* Iterator Pattern: Manages collections of games and teams 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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | - Open-source, customizable, strong performance and scalability.  - Cost-effective, high stability, security, extensive web server support.  - Technical expertise required, limited commercial support.  - Generally free, enterprise versions may have support costs. | - Unix-based, user-friendly interface, strong integration with Apple products.  - High performance, stability, excellent development tool support.  - Expensive hardware requirements, limited flexibility.  - Higher due to Apple hardware and software licenses. | - Widely used, extensive commercial support, strong compatibility with enterprise apps.  - User-friendly, wide range of development tools, strong web server support.  - Higher vulnerability to security threats, higher licensing costs.  - Higher due to Windows Server licenses and CALs. | - Native apps offer superior performance and integration.  - Web apps are accessible via browsers but may have limited functionality.  - Native app development requires platform-specific knowledge.  - Varies depending on the tools and platforms used. |
| **Client Side** | - Modern web browsers support HTML5, CSS3, JavaScript.  - Platform-independent, consistent user experience.  - Requires thorough testing for compatibility, performance varies.  - Ensure responsive design principles, use frameworks like Bootstrap. | - Modern web browsers support HTML5, CSS3, JavaScript.  - Platform-independent, consistent user experience.  - Requires thorough testing for compatibility, performance varies.  - Ensure responsive design principles, use frameworks like Bootstrap. | - Modern web browsers support HTML5, CSS3, JavaScript.  - Platform-independent, consistent user experience.  - Requires thorough testing for compatibility, performance varies.  - Ensure responsive design principles, use frameworks like Bootstrap. | - Native apps require Swift (iOS), Kotlin/Java (Android).  - Web apps are easier to maintain and update.  - Native apps require platform-specific knowledge.  - Consider cross-platform frameworks like React Native, Flutter. |
| **Development Tools** | - **Languages**: Python, Java, C++, PHP, JavaScript.  - **IDEs**: Eclipse, IntelliJ IDEA, Visual Studio Code.  - Open-source tools reduce costs, may require setup and maintenance.  - Generally free, some enterprise tools may have fees. | - **Languages**: Swift, Objective-C, Java, Python.  - **IDEs**: Xcode, Eclipse, IntelliJ IDEA.  - Requires expertise in Apple-specific tools, higher hardware costs.  - Xcode is free, Apple Developer Program costs $99/year. | - **Languages**: C#, Java, Python, JavaScript.  - **IDEs**: Visual Studio, Eclipse, IntelliJ IDEA.  - Widely used tools, extensive support.  - Licensing costs for certain development tools. | - **Languages**: Swift (iOS), Kotlin/Java (Android), JavaScript (React Native), Dart (Flutter).  - **IDEs**: Xcode, Android Studio, Visual Studio Code.  - Cross-platform tools streamline development.  - Varies, some free options, others paid licenses. |

##### Server-Side Platforms

1. **Linux:**

* **Characteristics:**
  + Open-source and highly customizable.
  + Strong performance and scalability.
  + Wide community support.
* **Advantages:**
  + Cost-effective due to being open-source.
  + High stability and security.
  + Extensive support for web server technologies (e.g., Apache, Nginx).
* **Weaknesses:**
  + Requires technical expertise to set up and maintain.
  + Limited commercial support compared to Windows and Mac.
* **Licensing Costs:**
  + Generally free, but enterprise versions (e.g., Red Hat) may have support costs.

1. **Mac:**

* **Characteristics:**
  + Unix-based system with a user-friendly interface.
  + Strong integration with other Apple products and services.
* **Advantages:**
  + High performance and stability.
  + Excellent support for development tools (e.g., Xcode).
* **Weaknesses:**
  + Expensive hardware requirements.
  + Limited flexibility compared to Linux.
* **Licensing Costs:**
  + Higher due to the cost of Apple hardware and possible software licenses.

1. **Windows:**

* **Characteristics:**
  + Widely used with extensive commercial support.
  + Strong compatibility with various enterprise applications.
* **Advantages:**
  + User-friendly with a wide range of development tools (e.g., Visual Studio).
  + Strong support for web server technologies (e.g., IIS).
* **Weaknesses:**
  + Higher vulnerability to security threats.
  + Licensing costs for both the operating system and additional software.
* **Licensing Costs:**
  + Higher due to the cost of Windows Server licenses and CALs (Client Access Licenses).

##### Client-Side Platforms

1. **Web Browsers (Linux, Mac, Windows):**

* **Characteristics:**
  + Modern web browsers (Chrome, Firefox, Safari, Edge) support HTML5, CSS3, and JavaScript.
  + Responsive design ensures compatibility across different devices.
* **Advantages:**
  + Platform-independent, accessible via any web browser.
  + Consistent user experience across different operating systems.
* **Weaknesses:**
  + Requires thorough testing to ensure compatibility.
  + Performance may vary depending on the browser and device.
* **Software Development Considerations:**
  + Ensure the use of responsive design principles.
  + Test extensively on different browsers and devices.
  + Consider using frameworks like Bootstrap or Foundation for responsive design.

1. **Mobile Platforms (iOS, Android):**

* **Characteristics:**
  + Native apps provide the best performance and integration with device features.
  + Web apps can be accessed via mobile browsers but may have limited functionality compared to native apps.
* **Advantages:**
  + Native apps offer superior performance and user experience.
  + Web apps are easier to maintain and update.
* **Weaknesses:**
  + Developing native apps requires knowledge of platform-specific languages (Swift for iOS, Kotlin/Java for Android).
  + Web apps may not fully utilize device capabilities.
* **Software Development Considerations:**
  + Native app development requires expertise in platform-specific languages and tools.
  + Web apps require responsive design to ensure compatibility with different screen sizes.
  + Consider using cross-platform frameworks (e.g., React Native, Flutter) for native app development.

##### Development Tools

1. **Linux:**

* **Programming Languages and Tools:**
  + Languages: Python, Java, C++, PHP, JavaScript.
  + IDEs: Eclipse, IntelliJ IDEA, Visual Studio Code.
* **Impact on Development Team:**
  + May require multiple development teams specializing in different tools and languages.
  + Open-source tools can reduce costs but may require more setup and maintenance.
* **Licensing Costs:**
  + Generally free, but some enterprise tools may have licensing fees.

1. **Mac:**

* **Programming Languages and Tools:**
  + Languages: Swift, Objective-C, Java, Python.
  + IDEs: Xcode, Eclipse, IntelliJ IDEA.
* **Impact on Development Team:**
  + Development for Apple products requires expertise in Apple-specific tools and languages.
  + Higher hardware costs for development.
* **Licensing Costs:**
  + Xcode is free, but the Apple Developer Program costs $99/year for app distribution.

1. **Windows:**

* **Programming Languages and Tools:**
  + Languages: C#, Java, Python, JavaScript.
  + IDEs: Visual Studio, Eclipse, IntelliJ IDEA.
* **Impact on Development Team:**
  + Widely used tools with extensive support and documentation.
  + Licensing costs for certain development tools.
* **Licensing Costs:**
  + Visual Studio has both free and paid versions, with licensing costs for enterprise editions.

1. **Mobile Devices:**

* **Programming Languages and Tools:**
  + Languages: Swift (iOS), Kotlin/Java (Android), JavaScript (React Native), Dart (Flutter).
  + IDEs: Xcode (iOS), Android Studio (Android), Visual Studio Code.
* **Impact on Development Team:**
  + Cross-platform development tools can streamline the process but may have limitations.
  + Native app development requires expertise in platform-specific languages and tools.
* **Licensing Costs:**
  + Varies depending on the tools and platforms used, with some free options and others requiring paid licenses.

## Recommendations

1. **Operating Platform**: Linux is suggested as the main operating platform for The Gaming Room’s "Draw It or Lose It" web application. Known for its stability, security, and scalability, Linux is ideal for hosting web applications. Moreover, being an open-source platform, Linux is cost-effective and reduces licensing expenses. The extensive community support and documentation available for Linux will also aid in smooth development and troubleshooting.
2. **Operating Systems Architectures**: The proposed architecture for Linux features kernel-level memory management and process scheduling. Linux’s monolithic kernel efficiently manages system resources, which is crucial for a high-performance web application. This architecture leverages Linux’s ability to handle multiple processes concurrently, ensuring seamless scalability as the user base grows. Additionally, utilizing containerization technologies like Docker can further enhance scalability and resource management by isolating application components.
3. **Storage Management**: A robust database management system such as MySQL or PostgreSQL is essential for managing the game’s data. These databases are known for their high reliability, performance, and scalability, making them suitable for the game’s needs. They can efficiently manage user information, game state, and other critical data, providing the necessary performance and scalability to support a large number of concurrent users.
4. **Memory Management**: Linux employs advanced memory management techniques to efficiently handle the application's memory requirements. Techniques such as paging, swapping, and virtual memory ensure the application runs smoothly even under heavy load conditions. Implementing caching mechanisms like Redis or Memcached can also enhance performance by reducing database load and speeding up data retrieval processes.
5. **Distributed Systems and Networks**: To facilitate communication between various platforms, RESTful APIs are recommended. These stateless APIs can easily manage requests from different clients, including web browsers and mobile devices. Adopting a microservices architecture can further improve the application’s modularity, making it easier to update and scale individual components. Additionally, deploying the application on cloud platforms like AWS or Azure provides flexibility, scalability, and various networking services to ensure efficient communication and data transfer between distributed components.
6. **Security**: Security is critical for protecting user information and maintaining trust in the application. Implementing secure communication protocols such as HTTPS will encrypt data transmitted between clients and the server, preventing eavesdropping and tampering. Robust authentication mechanisms, like OAuth 2.0 or JWT (JSON Web Tokens), should be used to verify user identities. Conducting regular security audits and penetration testing can help identify and mitigate vulnerabilities. Furthermore, employing firewall rules, intrusion detection systems (IDS), and regular software updates will enhance the application's overall security.