

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/28/2024 | Shawn Henly | Initial creation of the software design document. |

**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](#_heading=h.35nkun2)

This software design document outlines the development of 'Draw It or Lose It,' a web-based gaming application designed to offer a unique, interactive gaming experience. The platform aims to integrate advanced features like real-time multiplayer capabilities, cross-platform support, and customizable game rooms. This document addresses the key design considerations, including system architecture, domain models, and security measures, ensuring a scalable and user-friendly gaming environment.

## Requirements

The project aims to develop a responsive and scalable gaming platform with real-time multiplayer support. Key requirements include cross-platform compatibility (Windows, macOS, Linux, and mobile devices), user account management, in-game chat functionality, high-performance graphics rendering, and robust security measures to protect user data.

## [Design Constraints](#_heading=h.1ksv4uv)

The primary design constraints include ensuring seamless operation in a web-based distributed environment, compatibility across different operating systems and devices, scalability to handle a large number of concurrent users, and adherence to strict security protocols to safeguard user information.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

The UML class diagram includes classes such as User, GameRoom, GameSession, and Chat. 'User' class represents the gamers with attributes like username and password. 'GameRoom' class encapsulates the details of each gaming session. 'GameSession' links users to the active game, and 'Chat' enables in-game communication. The model uses inheritance and association, demonstrating encapsulation and polymorphism to efficiently meet software requirements. With these requirements, we can prove not only to ourselves, but to those around us what the true meaning of these things are. We can in essence prove to the world how these things operate and how

**"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](#_heading=h.z337ya)

1. **Classes and Hierarchies**:
   * At the top level, there's a ProgramDriver class interacting with a SingletonTester. This indicates the application's entry point, responsible for initializing and testing the singleton pattern implementation.
   * The SingletonTester class is used to ensure that the GameService class is correctly implemented as a singleton.
   * The GameService class acts as a central point for managing games. It contains attributes like games, nextGamesId, nextPlayerId, nextTeamId, and a static reference to itself (service), which is a characteristic of the singleton pattern.
   * Lower in the hierarchy are the Game, Team, and Player classes, each representing different entities within the game application. They are connected to GameService with associations indicating a relationship.
2. **Singleton Pattern**:
   * The GameService class is implemented as a singleton, indicated by its static getInstance method. This ensures that only one instance of GameService is created, providing a global point of access to the game management functionality.
3. **Composition and Aggregation**:
   * The diagram shows a composition relationship between Game and Team, and between Team and Player. It portrays that Game objects contain Team objects, and Team objects contain Player objects.
   * These relationships indicate a strong life-cycle dependency. For example, when a Game object is destroyed, its associated Team objects would also cease to exist.
4. **Inheritance**:
   * The Entity class is a base class for Game, Team, and Player. It holds common attributes (id and name) and behaviors (getId, getName, toString methods), demonstrating inheritance.
   * This design promotes code reuse and simplifies maintenance. The common properties and methods of game entities are centralized in the Entity class, from which other specific classes derive.
5. **Encapsulation**:
   * Each class encapsulates its data and behavior. For instance, GameService encapsulates game management logic, Game encapsulates details about a game, and so on. This encapsulation ensures that the internal state of objects is protected and only accessible through their methods.
6. **Method Definitions**:
   * Classes like GameService, Game, Team, and Player include methods that are likely to manipulate their respective data. For example, addGame, addTeam, and addPlayer suggest functionality to add new entities.
7. **List Usage**:
   * The use of List<Game>, List<Team>, and List<Player> indicates an understanding of modern collections frameworks, which are essential for handling multiple objects efficiently.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Characteristics: MacOS servers are known for their stable and secure environment. They offer excellent support for a range of web technologies and have robust in-built features for performance optimization.  Advantages: High security, reliable performance, and strong support for multimedia applications make MacOS servers suitable for web-based applications.  Weaknesses: Limited market share compared to Windows and Linux, resulting in a potentially smaller user base. Higher cost is also a factor. | Characteristics: Linux servers are highly customizable, scalable, and known for their stability. They support a wide range of programming languages and technologies.  Advantages: Open-source nature reduces costs and allows extensive customization. It's widely used for web servers and has a large community for support.  Weaknesses: Can require more technical expertise to manage and optimize, and the diversity of distributions can lead to compatibility issues. | Characteristics: Windows servers are user-friendly and support a wide range of enterprise applications and services.  Advantages: Easy integration with other Windows-based systems and software. Broad support for various web technologies and strong enterprise support.  Weaknesses: Can be more vulnerable to security threats compared to Linux. Licensing costs are also a consideration. | Characteristics: Mobile device servers are emerging as alternatives for certain types of lightweight, distributed applications.  Advantages: Provide the flexibility to host applications directly on mobile devices, enhancing accessibility and potentially reducing latency.  Weaknesses: Limited processing power and storage compared to traditional servers, and potentially less reliable connectivity. |
| **Client Side** | Development for Mac clients often requires specific expertise in Apple's ecosystem, including Swift or Objective-C for native applications. The development process can be costlier and time-intensive due to the need for specific hardware and software tools. | Linux client development is diverse due to the variety of distributions. This requires broad compatibility testing, potentially increasing time and cost. Open-source tools can reduce direct costs but may require more specialized expertise. | Windows client development is generally well-supported with a range of development tools and libraries. Costs can be moderate, but the platform's popularity means there's a wealth of expertise available, potentially reducing development time. | Developing for mobile clients involves considerations of different operating systems (iOS, Android), varied screen sizes, and performance constraints. Costs and time can vary significantly based on the application's complexity. Expertise in mobile-specific languages and frameworks is essential. |
| **Development Tools** | Xcode is the primary IDE, with Swift and Objective-C being the main programming languages. Other tools include AppCode, and for web development, standard web technologies like HTML, CSS, JavaScript, and server-side languages like PHP, Python, or Ruby can be used. | A wide range of IDEs including Eclipse, IntelliJ IDEA, and others. Common languages are Python, PHP, Java, and C/C++. For web development, the LAMP (Linux, Apache, MySQL, PHP/Python/Perl) stack is prevalent. | Visual Studio is a primary IDE for Windows development. Languages include C#, VB.NET, and C++. For web development, the ASP.NET framework is commonly used along with HTML, CSS, and JavaScript. | For iOS, Xcode with Swift or Objective-C. For Android, Android Studio with Java or Kotlin. Cross-platform development can be done using frameworks like React Native, Flutter, or Xamarin using languages like JavaScript (React Native) and Dart (Flutter). |

## 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**: Based on the analysis, a cross-platform approach using web technologies like HTML5 and JavaScript frameworks is recommended. This allows for seamless operation across Windows, macOS, Linux, and mobile platforms.
2. **Operating Systems Architectures**: A microservices architecture is recommended to facilitate scalability and ease of maintenance across different operating systems.
3. **Storage Management**: Cloud-based storage solutions like Amazon S3 or Google Cloud Storage are recommended for their scalability, reliability, and security.
4. **Memory Management**: Efficient memory management will be achieved through optimized coding practices and leveraging built-in garbage collection mechanisms in the chosen development languages.
5. **Distributed Systems and Networks**: The application will use RESTful APIs for server-client communication, ensuring effective interaction across different platforms. Network reliability and latency considerations will be addressed through CDN deployment and efficient data transmission protocols.
6. **Security**: Implement robust encryption for data transmission (SSL/TLS), use OAuth for user authentication, and ensure regular security audits to protect user information across platforms.

**Project Two**

Server Side Evaluation

**Linux**

* **Characteristics**: Linux offers extensive support for server-based deployments, making it a popular choice for web servers. It is known for its stability, security, and flexibility in hosting web applications.
* **Advantages**: High configurability, open-source nature (leading to no licensing fees for the OS), and strong community support. Excellent for scalable web applications.
* **Weaknesses**: Requires technical expertise to manage and configure. Less direct support compared to commercial alternatives.
* **Licensing Costs**: Generally, there are no licensing costs for using Linux as a server operating system, though support services from distributions like Red Hat may incur fees.

**Mac**

* **Characteristics**: Mac OS, with its Unix-based foundation, can serve as a reliable platform for web-based applications, though less commonly used than Linux for servers.
* **Advantages**: Stable and secure environment with an intuitive interface. Integrated development environment for Apple-related applications.
* **Weaknesses**: Limited hardware compatibility and higher initial cost. Not typically used in server environments, leading to potentially less community support for this use case.
* **Licensing Costs**: Mac OS comes with Apple hardware without separate licensing fees for the OS, but the initial hardware investment is higher.

**Windows**

* **Characteristics**: Windows Server offers a user-friendly environment with extensive support for web-based applications, including native support for .NET applications.
* **Advantages**: Easy integration with other Windows-based environments and services. Strong support and documentation.
* **Weaknesses**: Higher licensing costs. Can be more susceptible to security vulnerabilities than Linux.
* **Licensing Costs**: Licensing costs for Windows Server vary based on the version and licensing model chosen, such as CAL (Client Access License) or Core-based licensing.

#### Client Side Evaluation

* **Requirements**: Development must accommodate multiple client types (iOS, Android, Linux, Mac, Windows) with a modern, responsive HTML interface.
* **Considerations**: Ensuring compatibility across all web browsers and mobile devices necessitates a responsive design framework (e.g., Bootstrap, Foundation) and testing on multiple devices and browsers. This can increase development time and costs.
* **Development Tools**: Use of cross-platform development tools and languages (e.g., HTML5, CSS3, JavaScript) is essential. For mobile, frameworks like React Native or Flutter can facilitate development across iOS and Android with a single codebase.

#### Development Tools Evaluation

* **Programming Languages**: HTML5, CSS3, JavaScript for front-end; server-side languages can vary (PHP, Python, Ruby, Node.js) based on the server environment.
* **IDEs and Tools**: Visual Studio Code, IntelliJ IDEA, and Xcode for macOS and iOS development. Android Studio for Android development. Docker for containerization and consistent environments.
* **Impact on Development Team**: Requires a diverse skill set to manage multiple languages and platforms. Depending on the project's scope, separate teams for front-end, back-end, and mobile development may be necessary.
* **Licensing Costs**: Most development tools and languages mentioned are open-source and free to use. However, proprietary tools or IDEs, and the need for mobile development accounts (e.g., Apple Developer Program) can incur costs.

**Project Three**

## 

## 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**:

In considering the expansion of "Draw It or Lose It," a cloud-based solution such as Amazon Web Services (AWS), Google Cloud, or Microsoft Azure is advisable. AWS, in particular, appears most suitable due to its specialized services like AWS GameLift, which are tailored for enhancing the performance and scalability of gaming applications.

1. **Operating Systems Architectures**:

AWS supports multiple operating systems, offering flexibility for development environments. This compatibility with various systems, including Linux and Windows Server, allows for a diverse development approach, aligning with our project's requirements and facilitating a smoother game development and deployment process.

1. **Storage Management**:

Utilizing Amazon S3 offers a dependable and expandable framework for the preservation of gaming assets and participant information. Coupling this with Amazon RDS or DynamoDB enhances the system's capability to oversee and conserve structured information effectively. This integration is essential for ensuring the system's resilience and its ability to scale, pivotal elements for upholding the fidelity of the game and engaging the user base actively.

1. **Memory Management**:

Efficient memory management is crucial for supporting extensive user interaction and gameplay. AWS provides configurable memory resources, enabling optimized performance for varying workload demands. The use of services like ElastiCache further enhances data retrieval speeds, contributing to a smoother user experience.

1. **Distributed Systems and Networks**:

The global infrastructure of AWS facilitates seamless communication and content delivery across different geographic locations. Implementing services such as Elastic Load Balancing and leveraging AWS’s networking capabilities ensures reliable game performance and availability, critical for maintaining a consistent and enjoyable player experience.

1. **Security**:

Security is of utmost importance, particularly in the management of confidential player details. AWS delivers an extensive suite of security functionalities, encompassing AWS Identity and Access Management (IAM), AWS Shield for defense against DDoS attacks, and Amazon Cognito for user verification. These mechanisms establish a fortified infrastructure, safeguarding user information and maintaining the integrity of the gaming ecosystem.