

Draw It or Lose It

# **CS 230 Project Software Design**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 06/21/2024 | Brianna Reed | Initial draft of additional recommendations section for Project Three. Included detailed analysis of operating platforms, system architectures, storage management, memory management, distributed systems, networks, security requirements, scalability, continuous integration/continuous deployment (CI/DI), monitoring and logging, and user experience (UX) optimization for the expansion of the Draw It or Lose It game. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based version of their existing Android game, Draw It or Lose It, to reach a wider audience and support multiple platforms. The game allows one or more teams to participate, with each team consisting of multiple players. Critical requirements include ensuring unique names for games and teams, as well as maintaining a single instance of the game in memory at any given time. This document outlines the software design for the web-based game, addressing key design constraints and presenting a domain model to facilitate the development process.

## Requirements

The Gaming Room's requirements are to develop a web-based game application that:

* Supports multiple teams within a game
* Allows multiple players per team
* Ensures unique names for games and teams
* Guarantees only one instance of the game exists in memory at any time

## [Design Constraints](#_2et92p0)

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

1. **Scalability**: The application must handle multiple concurrent users, requiring efficient resource management and load balancing across servers.
2. **Consistency**: To maintain data consistency, especially for game and team names, a robust database management system is necessary.
3. **Performance**: The application should render images and process user inputs with minimal latency, necessitating optimized algorithms and efficient data structures.
4. **Security**: Protecting user data and ensuring secure communication between clients and servers is paramount, requiring encryption and secure authentication mechanisms.

These constraints will influence the choice of technologies, architectural patterns, and implementation strategies.

## [System Architecture View](#_ilbxbyevv6b6)

This section serves as a placeholder for future projects. While not required for this specific project, describing the system and subsystem architecture, including physical components or tiers, and providing a logical topology of communication and storage, is essential for understanding the overall architecture in other projects.

## 

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram represents the domain model for the game application. Key classes include:

* **Entity**: A base class for common attributes (id, name) and behaviors.
* **GameService**: A singleton class managing game instances, ensuring only one instance exists in memory.
* **Game**: Inherits from Entity, representing a game with a list of teams.
* **Team**: Inherits from Entity, representing a team with a list of players.
* **Player**: Inherits from Entity, representing a player.
* **ProgramDriver**: The entry point of the application.
* **SingletonTester**: A class to test the singleton behavior of GameService.

Object-oriented programming principles such as inheritance (Entity as the base class) and design patterns (singleton and iterator) are utilized to ensure code reusability, maintainability, and efficient management of game entities.

**"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** | **Characteristics:**  -Unix-based OS known for stability and robust performance  -Limited options for server software compared to Linux  **Advantages:**  -High security and user-friendly interface  -Excellent graphics support, beneficial for game rendering  **Weaknesses:**  -High licensing costs for both OS and hardware  -Smaller market share for server-side applications  **Deployment Method:**  -Supports server-based deployment using Apache or Nginx  -Less common in server environments but viable  **Licensing Costs:**  -High due to proprietary nature and hardware requirements  **Evaluation:** Mac systems are known for their stability and robust performance, making them suitable for hosting web-based applications. They offer a user-friendly interface and excellent graphics support, which can be beneficial for a game like Draw It or Lose It. However, Macs are generally more expensive than other platforms, which could increase overall development and maintenance costs. Additionally, the server-side market share for macOS is relatively small, which might limit the availability of macOS-specific server tools and resources. | **Characteristics:**  -Open-source, highly customizable, and secure  -Supports various web servers like Apache, Nginx, and Lighttpd  **Advantages:**  -No licensing fees, reducing costs  -High stability and performance with extensive community support  **Weaknesses:**  -Steeper learning curve for developers unfamiliar with Linux  **Deployment Method:**  -Robust server-based deployment options, ideal for scaling  -Widely used for hosting web applications  **Licensing Costs:**  -None (open-source)  **Evaluation:** Linux is highly regarded for its performance, security, and flexibility. It is open-source, which reduces costs associated with software licensing. Linux servers are known for their stability and can handle high-traffic loads efficiently, making them an excellent choice for hosting a web-based game. The large community support and extensive documentation available for Linux also make it easier to find solutions to any issues that arise. However, Linux may require a steeper learning curve for developers unfamiliar with it. | **Characteristics:**  -Widely used OS with broad software compatibility  -Supports IIS (Internet Information Services) and other web servers  **Advantages:**  -Familiar interface for many developers  -Strong enterprise support and extensive documentation  **Weaknesses:**  -Higher licensing costs and more vulnerable to security threats  **Deployment Method:**  -Robust server-based deployment with IIS  -Can handle large-scale user bases effectively  **Licensing Costs:**  -High due to proprietary software and server licensing fees  **Evaluation:** Windows servers are widely used and supported, making them a reliable choice for hosting web-based applications. They offer good compatibility with a wide range of software and development tools, and many developers are already familiar with the Windows environment. However, Windows servers tend to be more vulnerable to security threats compared to Linux servers, requiring more robust security measures. Additionally, licensing costs for Windows servers can add to the overall expenses. | **Characteristics:**  -Dominant mobile platforms with extensive user bases  -Require separate development environments (Xcode for iOS, Android Studio for Android)  **Advantages:**  -Wide reach and strong ecosystems  -Distribution through app stores  **Weaknesses:**  -Separate codebases and development expertise needed  -High competition in app stores  **Deployment Method:**  -Mobile devices act as clients, not servers  -Require optimization for server interaction  **Licensing Costs:**  -Developer accounts needed for app store distribution (iOS: $99/year, Android: $25 one-time)  **Evaluation:** Hosting web-based applications on mobile devices is generally not practical due to the limited processing power and storage capacity compared to traditional servers. However, mobile devices can serve as clients accessing the game hosted on other platforms. For client-side development, considerations include ensuring compatibility with various mobile operating systems (iOS, Android) and optimizing performance to handle the game’s graphics and network interactions efficiently. |
| **Client Side** | **Development Considerations:**  -Higher cost of Apple devices  -Need for expertise in macOS-specific development tools like Xcode  **Development Tools:**  -Xcode, Swift, Objective-C  **Cost and Time:**  -High initial cost and time for training or hiring developers  **Advantages:**  -High-quality graphics and user experience  -Enhances gaming experience  **Challenges:**  -Ensuring cross-platform compatibility  **Evaluation:** Developing for Mac clients involves considerations like the higher cost of Apple devices and the need for expertise in macOS-specific development tools like Xcode. The cost and time to train developers or hire those with macOS expertise must be considered. Macs are known for their high-quality graphics and smooth user experience, which can enhance the gaming experience for users. Additionally, ensuring cross-platform compatibility will be crucial for a seamless user experience across different devices. | **Development Considerations:**  -Open-source nature reduces software licensing costs  -Popular tools: Eclipse, IntelliJ IDEA  **Development Tools:**  -Java, Python, C++, Eclipse, IntelliJ IDEA  **Cost and Time:**  -Lower costs but requires ensuring compatibility with various distributions  **Advantages:**  -Cost-effective and flexible  **Challenges:**  -Smaller market share among desktop users  **Evaluation:** Supporting Linux clients can be cost-effective due to the open-source nature of Linux, reducing software licensing costs. Development tools like Eclipse and IntelliJ IDEA are popular on Linux. However, Linux has a smaller market share among desktop users, so the cost and time to support Linux clients must be balanced against the potential user base. Ensuring that the game runs smoothly on various Linux distributions can also add complexity to the development process. | **Development Considerations:**  -Most popular desktop OS, essential for broad user reach  -Wide range of development tools: Visual Studio  **Development Tools:**  -C#, .NET, Visual Studio  **Cost and Time:**  -Consider development and maintenance costs  -Ensure compatibility with different Windows versions  **Advantages:**  -Familiar environment for many developers  **Challenges:**  -More prone to security vulnerabilities  **Evaluation:** Windows is the most popular desktop operating system, which means supporting Windows clients is essential for reaching a broad user base. Development for Windows is straightforward with a wide range of tools available, such as Visual Studio. However, the cost and time to develop and maintain a Windows client must be considered, along with ensuring compatibility with different versions of Windows. Windows devices are also more prone to security vulnerabilities, necessitating robust security measures. | **Development Considerations:**  -Diverse range of devices and operating systems  -Separate development tools: Xcode for iOS, Android Studio for Android  **Development Tools:**  -Swift for iOS, Java/Kotlin for Android  **Cost and Time:**  -High cost and time for developing, testing, and maintaining  **Advantages:**  -Broad mobile audience reach  **Challenges:**  -Handling varying screen sizes and hardware capabilities  **Evaluation:** Developing for mobile clients involves considering the diverse range of devices and operating systems, primarily iOS and Android. Each platform has its own development tools (Xcode for iOS, Android Studio for Android) and guidelines. The cost and time to develop, test, and maintain the game across multiple devices can be significant. Expertise in mobile development is essential to optimize performance and ensure a smooth user experience. Additionally, the app must be designed to handle varying screen sizes and hardware capabilities. |
| **Develent Tools** | **Tools:**  -Xcode, Swift, Objective-C  **Impact on Development Team:**  -Requires proficiency in macOS development tools  -Extensive documentation and developer support  **Licensing Costs:**  -High cost of Apple hardware and software licenses  **Evaluation:** Development for Mac typically involves using Xcode, Apple’s integrated development environment (IDE). Xcode provides a comprehensive suite of tools for developing, testing, and debugging macOS applications. The development process benefits from Apple's extensive documentation and developer support. However, developers need to be proficient in Swift or Objective-C, which are the primary programming languages for macOS development. The cost of Apple hardware and software licenses should also be factored into the project budget. | **Tools:**  -Eclipse, IntelliJ IDEA, Java, Python, C++  **Impact on Development Team:**  -Flexible and customizable development environment  -Strong community support and extensive documentation  **Licensing Costs:**  -Reduced costs due to open-source nature  **Evaluation:** Linux supports a wide range of development tools, including open-source IDEs like Eclipse and IntelliJ IDEA. Developers can use various programming languages, such as Java, Python, and C++. The flexibility and customization options available in Linux can streamline the development process. Additionally, the strong community support and extensive documentation can help resolve any issues quickly. The open-source nature of Linux reduces costs, but developers need to be proficient in using Linux environments. | **Tools:**  -Visual Studio, C#, .NET  **Impact on Development Team:**  -Comprehensive environment for coding, debugging, and testing  -Wide availability of resources and support  **Licensing Costs:**  -High due to proprietary software  **Evaluation:** Windows development is supported by robust tools like Visual Studio, which provides a comprehensive environment for coding, debugging, and testing applications. Developers can use languages such as C#, .NET, and C++. The wide availability of resources and support for Windows development can accelerate the development process. However, licensing costs for Windows software and tools should be considered. Ensuring compatibility with various Windows versions also adds to the complexity. | **Tools:**  -Xcode for iOS, Android Studio for Android  **Impact on Development Team:**  -Requires expertise in platform-specific tool  -Ensuring optimal performance and user experience on diverse devices  **Licensing Costs:**  -Developer accounts (iOS: $99/year, Android: $25 one-time)  **Evaluation:** Developing for mobile devices involves using platform-specific tools like Xcode for iOS and Android Studio for Android. Both environments offer extensive support for coding, testing, and debugging mobile applications. Developers must be skilled in Swift for iOS and Java/Kotlin for Android. The cost and time to develop and maintain applications for both platforms can be high, but this is necessary to reach a broad mobile audience. Ensuring optimal performance and user experience on diverse devices is crucial.Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices. |

## 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**: Recommend deploying the game on a Linux-based server for its cost-effectiveness, security, and scalability. Linux servers are highly stable and can handle high traffic loads efficiently, making them suitable for hosting a web-based game.
2. **Operating Systems Architectures**: Linux provides a robust and flexible architecture that supports various web technologies and frameworks. Its open-source nature allows for customization and optimization according to specific project needs, ensuring high performance and reliability.
3. **Storage Management**: Implement a cloud-based storage solution like Amazon S3 or Google Cloud Storage to ensure scalability and redundancy. These services offer robust data management and backup solutions, essential for handling the game's data securely and efficiently.
4. **Memory Management**: Linux servers efficiently manage memory using techniques like paging and swapping, ensuring optimal performance. Utilizing memory management tools and monitoring systems will help maintain the game's performance and prevent memory leaks.
5. **Distributed Systems and Networks**: To facilitate communication between various platforms, implement a RESTful API architecture. This will allow seamless interaction between clients and the server, ensuring data consistency and synchronization. Utilize load balancers and redundant network paths to maintain connectivity and handle outages.
6. **Security**: Security is paramount for protecting user data. Implement encryption for data at rest and in transit, use secure authentication mechanisms like OAuth, and conduct regular security audits. Linux's inherent security features, combined with additional security measures, will protect user information and ensure compliance with data protection regulations.
7. **Scalability:** Design the game’s architecture to support horizontal scaling. This involves adding more servers to handle increased load rather than upgrading existing servers. Containerization technologies like Docker can help in managing and deploying multiple instances of the game efficiently across various servers.
8. **Continuous Integration/Continuous Deployment (CI/CD):** Implement a CI/CD pipeline to automate the testing and deployment process. This will enable rapid development cycles, ensuring that new features and updates can be released frequently without disrupting the game’s performance. Tools like Jenkins, GitLab CI, or CircleCI can be used to set up CI/CD pipelines.
9. **Monitoring and Logging:** Deploy comprehensive monitoring and logging solutions to track the performance and health of the game’s servers and applications. Tools like Prometheus for monitoring and ELK Stack (Elasticsearch, Logstash, Kibana) for logging can provide valuable insights into system performance, helping to identify and resolve issues promptly.
10. **User Experience (UX) Optimization:** Focus on optimizing the user experience by ensuring the game is responsive and performs well across different devices and network conditions. Techniques such as lazy loading for game assets, optimizing network requests, and minimizing the use of heavy graphics can improve the overall user experience. Regular user feedback and usability testing should be incorporated into the development process.