

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 | 03/24/2024 | Jose Medina | Consultation to a new client for web-based game. |
| 2.0 | 04/05/2024 | Jose Medina | Operating Platforms |
| 3.0 | 04/17/2024 | Jose Medina | 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)

Draw It or Lose It, The Gaming Room's popular Android app, is set to expand into a web-based application accessible across various platforms. This design document outlines the transition from a single-platform application to a multi-platform web application that will provide an interactive gaming experience for multiple teams and players. The proposed web application will support simultaneous gameplay, ensuring unique game sessions with distinctive identifiers for games, teams, and players. This document also addresses the design constraints, system architecture, domain model, and evaluates the application across different operating platforms.

## Requirements

Business Requirements:

Multi-platform Accessibility: The game must be accessible on various platforms, including web browsers and mobile devices, to reach a broader audience.

User Engagement: The ability to support multiple teams and an interactive gameplay experience is crucial to keep users engaged.

Brand Consistency: The transition from an Android app to a web-based game must maintain the brand's look and feel to provide a consistent user experience.

Monetization Strategy: Implementing features that support in-game purchases or ad placements to generate revenue.

Scalability: The game's infrastructure should support a growing number of users as the game gains popularity.

Technical Requirements:

1. Cross-Browser Compatibility: Ensure the game operates seamlessly across different web browsers.

2. Responsive Design: The game should provide an optimal experience across various devices with different screen sizes.

3. Real-time Interaction: Support real-time gameplay with minimal latency for drawing and guessing actions.

4. Unique Session Management: The system must manage unique game sessions, with distinctive identifiers for each game, team, and player to prevent conflicts.

5. Data Integrity and Security: Safeguard user data with secure communication channels and proper storage solutions.

6. Maintainability: The game's codebase should be easy to maintain and update, with clear documentation and adherence to coding standards.

These requirements are critical to the successful expansion of "Draw It or Lose It" into a multi-user, web-based gaming platform that provides a consistent and engaging experience for players across different computing environments.

## [Design Constraints](#_2et92p0)

Developing a web-based version of Draw It or Lose It presents several design constraints:

Compatibility: The application must function seamlessly across various web browsers and devices.

Concurrency: Support for multiple teams and players interacting in real-time.

Performance: Efficient rendering of stock drawings without latency.

Uniqueness: Unique identifiers and names for each game session, team, and player.

State Management: Ensuring game state is synchronized across all clients and the server.

The implications of these constraints include the need for a robust backend to handle multiple connections, a responsive frontend design, and a database design that prevents naming conflicts and ensures integrity.

## [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 "The Gaming Room" outlines the structure and relationships of the classes within the game application. Here's how the classes relate to each other and the object-oriented programming principles demonstrated:

Classes and Relationships:

1. Entity : Serves as the base class for `Game`, `Team`, and `Player` classes. It provides common attributes like `id` and `name`, which are inherited by the subclasses. The Entity class embodies the principle of reusability, avoiding duplication of these common attributes across each class.

2. GameService : This is a singleton class, which means there can only be one instance of this class at any time during the application's lifecycle. It manages a list of `Game` instances and provides methods to add and retrieve games. It demonstrates encapsulation, as the details of the game list and ID management are hidden from the other classes.

3. Game : Inherits from `Entity` and has a list of `Team` instances. Each game is uniquely identifiable by its `id` and `name`, and can contain multiple teams. This illustrates the use of aggregation, as a game aggregates teams.

4. Team : Also a subclass of `Entity`, a team has its own list of `Player` instances, indicating composition. A team is composed of players, and without a team, the players in this context do not exist. It has methods to add and retrieve players.

5. Player : Inherits from `Entity` as well, meaning it has an `id` and `name`. This shows inheritance and abstraction, where `Player` represents a specific type of entity with potentially more attributes and behaviors that are specific to a player in the game.

Object-Oriented Programming Principles:

1. Inheritance : Demonstrated by `Game`, `Team`, and `Player` classes inheriting from the `Entity` class. This allows for code reuse and a hierarchical classification.

2. Encapsulation : Exhibited by `GameService` controlling the access to game instances and providing methods to interact with the private games list.

3. Abstraction : `Entity` class provides an abstraction layer for common attributes. Other classes may expand on these attributes with more specific details.

4. Aggregation and Composition : `Game` uses aggregation to maintain a list of `Team` objects, and `Team` uses composition to manage `Player` objects. This reflects the whole-part relationships.

5. Singleton Pattern : `GameService` follows the Singleton pattern, ensuring it's the only class responsible for game management, which is crucial for maintaining a single state across the application.

Efficiency in Fulfilling Software Requirements:

* The Entity base class allows for extending common functionality easily and consistently across all entities.
* The Singleton instance of `GameService` ensures that games are managed centrally, providing a single source of truth and facilitating easy management of game states.
* The aggregation relationship between `Game` and `Team`, as well as the composition relationship between `Team` and `Player`, help manage the complexity of the game application by clearly defining ownership and life cycle dependencies among the objects.

By following these object-oriented principles and design patterns, the software design efficiently organizes the game's entities, ensures consistency of data and behavior, and facilitates maintainability and scalability of 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** | Characteristics and Advantages:  Mac servers are known for their robust security features and user-friendly interface. They integrate well with other Apple products and services, which could be advantageous for certain user demographics.  Weaknesses: There are no significant licensing cost for running a macOS Server, Hosting on Mac might be more expensive due to hardware costs and less common in server environments compared to Linux or Windows, potentially leading to higher operational costs and reduced availability of support. | Characteristics and Advantages:  Linux servers are widely used due to their stability, security, and performance. They are open-source, which can reduce costs and provide extensive customization options.  Weaknesses: Requires more technical expertise to set up and maintain. The variety of distributions can also introduce inconsistencies.  The potential licensing costs are generally very low or nonexistent because many Linux distributions are open-source and can be used freely. | Characteristics and Advantages:  Windows servers offer seamless integration with other Microsoft products and services. They provide a familiar environment for many administrators and support a broad range of enterprise applications.  Weaknesses:  Server requires purchasing a license, and the cost can vary depending on the edition. Licensing fees can be higher, and it may be more prone to security threats compared to Linux. | Characteristics and Advantages:  Mobile devices offer the advantage of portability and are increasingly used for web access. They can serve as portable game stations, reaching a broad audience.  Weaknesses: Limited by hardware capabilities, battery life, and the need for optimization to ensure responsiveness across various devices.  Mobile devices typically do not offer a traditional server-based deployment method for hosting websites in the way that dedicated server hardware and operating systems like Linux, Windows, or macOS do. |
| **Client Side** | Considerations:  The application should employ responsive web design for optimal display on various screens and be tested for cross-browser compatibility, including Safari, Chrome, Firefox, and Edge. The development should leverage modern front-end frameworks like React or Angular to handle inconsistencies and streamline responsive design. Performance optimization is crucial, focusing on minimizing load times and efficient data handling. A thorough testing strategy, covering unit, integration, and user acceptance tests across devices and browsers, is essential for identifying and resolving compatibility issues. Implementing continuous integration and deployment will facilitate automatic testing and deployment, ensuring consistent application performance across all platforms. | Considerations:  It’s crucial to account for the diversity of Linux distributions and desktop environments, which necessitates extensive testing to ensure application compatibility. Developers should focus on creating a responsive HTML interface that functions reliably across popular Linux browsers like Firefox, Chrome, and Opera. Utilizing cross-platform development tools and frameworks can help mitigate issues arising from the varied Linux ecosystem. Performance optimization and regular testing across different Linux distributions will be essential for maintaining a consistent user experience. Implementing a robust testing strategy, including automated and manual tests, will help identify and address compatibility issues efficiently. | Considerations:  Ensure compatibility with different Windows versions and popular browsers like Edge, Chrome, and Firefox. The varied hardware ecosystem of Windows necessitates thorough testing across multiple system configurations to guarantee performance and usability. Developers should prioritize creating a responsive HTML interface that adapts seamlessly to different screen sizes and resolutions common in the Windows environment. Utilizing cross-browser testing tools and responsive design frameworks will aid in addressing compatibility challenges. Regular performance optimization and compatibility testing across various Windows versions and hardware setups are critical for a smooth user experience. | Considerations:  For mobile devices operating systems like iOS and Android, development must prioritize touch interface design and optimization for smaller screens and mobile usage patterns. This entails refining UI/UX to ensure touch controls are intuitive and content is easily readable. The application should be tested across a range of devices to handle varying screen sizes, resolutions, and performance capabilities. Responsive design techniques are crucial for adapting the layout and functionality to fit mobile environments seamlessly. Performance optimization, focusing on fast load times and efficient data usage, is vital for enhancing the user experience on mobile networks. Regular updates and user feedback analysis will further refine the app, ensuring compatibility and a smooth operation across diverse mobile platforms. |
| **Development Tools** | Developing for the Mac operating system requires a team adept in using tools like Xcode for native applications, and web development tools such as Visual Studio Code, Atom, or Sublime Text for broader compatibility tasks. Proficiency in programming languages like Swift and Objective-C is essential for native development, while knowledge of cross-platform frameworks like React Native or Flutter can enhance the application's reach across different environments. Although Xcode is available at no cost, distributing through the Mac App Store necessitates an Apple Developer Program membership, incurring an annual fee. The complexity of ensuring compatibility across various browsers on macOS might necessitate multiple specialized development teams. Thus, effective project management is key to balancing the unique technical demands and development costs, ensuring the application performs optimally on macOS and other platforms. | Developing for the Linux operating system necessitates a development team proficient with a variety of IDEs and code editors such as Eclipse, IntelliJ IDEA, and Visual Studio Code, covering a wide range of programming languages from Java, Python, and C++ to web-centric ones like JavaScript, PHP, and Ruby. The diverse Linux ecosystem means that teams must be versatile, potentially requiring multiple specialized groups to handle different aspects of development, from native Linux applications to web-based solutions. Most development tools for Linux are open-source and free, eliminating direct licensing costs, but the complexity and variety of the Linux platform may necessitate investments in training and coordination across teams. | Developing for the Windows operating system, particularly with Visual Studio for .NET applications, requires a development team skilled in languages like C# and VB.NET for native apps, and HTML5, CSS, and JavaScript for web applications, using IDEs like Visual Studio and cross-platform tools like VS Code. The diverse technical requirements may necessitate multiple development teams specialized in different areas, such as native Windows development and web-based applications. While Visual Studio has licensing costs, especially for the professional editions, VS Code and other tools offer free versions that can mitigate overall expenses. | Developing for mobile devices necessitates using Android Studio for Android and Xcode for iOS, while cross-platform frameworks like React Native, Flutter, and Xamarin allow deployment on both platforms from a single codebase. This dual approach may require multiple development teams, one focusing on native platform-specific development and another on cross-platform solutions, to cover the broad spectrum of mobile technologies. Licensing costs are generally low, with Android Studio being free and Xcode requiring no direct cost, although iOS deployment necessitates an Apple Developer Program membership fee. |

## 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**: For The Gaming Room's needs, a cloud-based architecture using services like Amazon Web Services (AWS) or Google Cloud Platform (GCP) is recommended. I recommend utilizing Amazon Web Services as the operating platform. AWS provides a robust, scalable, and flexible infrastructure, making it an ideal choice for a game intended to expand across different computing environments. AWS supports a variety of operating systems and allows for the seamless integration of various services that can enhance the performance and scalability of the game (Amazon Web Services, Inc., n.d.).
2. **Operating Systems Architectures**: AWS's capability to support a variety of operating systems offers flexibility crucial for different technical requirements and user preferences. This includes support for widely used Linux distributions such as Amazon Linux, Ubuntu, and Red Hat, as well as Windows Server (Amazon Web Services, Inc., n.d.). Linux-based architectures are particularly valued for their efficiency, security, and compatibility with a wide range of server applications, scripting languages, and databases. This makes Linux a versatile choice for server-side operations.

On the client-side, to ensure broad compatibility and ease of access across different user devices, web technologies like HTML5, CSS, and JavaScript can be employed. These technologies are OS-agnostic and can run on any modern web browser, simplifying the deployment and maintenance of client applications across diverse operating systems. The combination of AWS’s flexible server-side capabilities and the use of universal client-side technologies ensures that developers can select an environment that optimally supports both the backend infrastructure and the frontend user experience.

1. **Storage Management**: For optimal storage management leveraging the storage solutions provided by AWS is what I would recommend. AWS offers Amazon S3 (Simple Storage Service) for scalable object storage, ideal for storing game assets like images and media files, and Amazon Elastic Block Store (EBS) for block storage, which is suitable for instances requiring persistent storage (Amazon Web Services, Inc., n.d.). These services not only ensure high availability and durability, critical for maintaining game integrity and user experience, but also support automated backups, scaling, and data redundancy.

In addition to these, a managed database service such as Amazon RDS can be utilized. Amazon RDS simplifies database setup, operations, and scaling, and is ideal for handling game states, user progress, settings, and high scores. The combination of AWS S3 for object storage, Amazon EBS for persistent block storage, and Amazon RDS for database management provides a comprehensive storage solution that ensures that game data is always available, secure, and efficiently managed.

1. **Memory Management**: For effective memory management utilizing AWS provides the necessary tools to ensure efficient resource utilization. AWS offers Amazon EC2 for scalable computing capacity, which allows for the dynamic allocation and management of computing resources based on the application's real-time needs (Amazon Web Services, Inc., n.d.). This capability is crucial for a game that demands high performance and responsiveness, as it facilitates the seamless scaling of resources without manual intervention.

In addition to Amazon EC2, AWS Elastic Beanstalk can also be leveraged for "Draw It or Lose It." Elastic Beanstalk is an orchestration service offered by AWS that automates the deployment, scaling, and management of applications. By using Elastic Beanstalk, the game can automatically adjust its computing resources in response to user demand. This not only ensures efficient memory utilization but also simplifies the management of the underlying infrastructure, allowing developers to focus more on game development and less on infrastructure management.

1. **Distributed Systems and Networks**: Employing a robust strategy for managing distributed systems and networks is critical to maintaining high availability and fault tolerance. AWS offers comprehensive network services that are instrumental in achieving this. AWS's Amazon Route 53 provides reliable DNS web services, and AWS Direct Connect establishes a dedicated network connection from premises to AWS, which can significantly reduce network costs and increase bandwidth throughput (Amazon Web Services, Inc., n.d.). These services ensure that the game remains accessible and performs consistently under varying network conditions.

In addition to these AWS services, implementing a microservices architecture can further enhance the game's operational efficiency and resilience. By decomposing the game into smaller, interconnected services that communicate through well-defined APIs, "Draw It or Lose It" can achieve more flexible and scalable interactions between different components of the game. This approach not only allows for effective load distribution across multiple instances using AWS’s load balancers but also provides redundancy to handle potential outages. The robust network infrastructure provided by AWS helps to mitigate connectivity issues, ensuring that each component of the distributed system communicates effectively, regardless of individual service loads or geographic distribution.

1. **Security:** To ensure comprehensive security AWS offers a suite of robust security

measures that can be integrated with best practices for data protection. AWS security services include data encryption both at rest and in transit, ensuring that all game data, whether stored or being transferred, is shielded from unauthorized access. Additionally, AWS provides detailed access controls and extensive monitoring capabilities through services like AWS Identity and Access Management and Amazon Cognito for user authentication and data synchronization (Amazon Web Services, Inc., n.d.). These features are crucial for protecting user information and maintaining the integrity of the game’s operations.

Building on this foundation, implementing HTTPS with Transport Layer Security is essential for secure data transmission, safeguarding data as it moves between clients and servers. Furthermore, using managed security services such as AWS Shield enhances protection against distributed denial of service (DDoS) attacks, offering another layer of security to the robust infrastructure. IAM policies play a critical role in controlling access to resources, ensuring that only authorized personnel can access sensitive information. Additionally, encryption-at-rest prevents unauthorized users from accessing stored data, providing a secure environment for user data and game assets.

Regular security audits and adherence to compliance standards, such as the General Data Protection Regulation, are also vital. These practices help to identify potential vulnerabilities and ensure that the game meets legal standards for user data protection, fostering trust and safety within the gaming community.

**Source**

Amazon Web Services, Inc. (n.d.). AWS documentation. Retrieved from   
https://aws.amazon.com/documentation/