

Kyle Marinaro

# **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 | 11/17/2024 | Kyle Marinaro | Preliminary draft detailing the design and implementation strategy. |

**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 has commissioned us to design a scalable and efficient software architecture to enhance the functionality of their web-based game, \*Draw It or Lose It\*. This report presents a comprehensive solution to develop a unified gaming application that adheres to object-oriented principles while leveraging the Singleton design pattern for optimal management of game data.

Our proposal ensures robust management of unique entities, including games, teams, and players, while satisfying critical client requirements such as compatibility across distributed platforms, high-level security measures, and efficient memory and storage management.

This design incorporates a domain model that aligns with UML specifications, facilitating maintainability and scalability for future feature enhancements. In addition, we will evaluate the suitability of various operating platforms—Linux, Mac, Windows, and Mobile—to establish a reliable and secure environment for hosting and running the application. Through this approach, we aim to create a sustainable, efficient, and user-centric gaming experience.

## Requirements

The Gaming Room has commissioned us to design a scalable and efficient software architecture aimed at enhancing the functionality of their web-based game, \*Draw It or Lose It\*. This report presents a comprehensive solution for developing a unified gaming application that adheres to object-oriented principles while utilizing the Singleton design pattern for optimal management of game data.

Our proposal ensures robust management of unique entities, including games, teams, and players, while fulfilling critical client requirements such as compatibility across distributed platforms, high-level security measures, and efficient memory and storage management.

This design incorporates a domain model that aligns with UML specifications, promoting maintainability and scalability for future feature enhancements. Additionally, we will evaluate the suitability of various operating platforms—Linux, Mac, Windows, and Mobile—to ensure optimal performance.

## [Design Constraints](#_2et92p0)

Developing a game application in a web-based distributed environment presents several key constraints that need careful consideration:

Platform Independence  
The application must be capable of running seamlessly across various operating systems. This requirement often necessitates the use of platform-agnostic frameworks, such as Java, or technologies that support containerization (like Docker), ensuring that the application can be deployed consistently regardless of the underlying environment.

Latency and Network Dependence

As a distributed system, the application's performance is inherently tied to network conditions. Variability in network speed, latency, and potential outages can significantly impact user experience. Developers need to implement strategies for handling these challenges, such as using asynchronous communication and optimizing data transfer protocols.

Security

With the transmission and storage of sensitive data, robust security measures are paramount. This includes employing encryption protocols for data in transit and at rest, implementing secure authentication mechanisms, and adhering to best practices for data protection to safeguard user information and maintain trust.

Scalability

The architecture must support scalability to accommodate fluctuating user loads and the expansion of game offerings. This requires a well-designed resource management strategy, allowing the system to efficiently allocate and deallocate resources based on demand, as well as incorporating load balancing and horizontal scaling solutions.

These constraints shape the development process by emphasizing the need for cross-platform compatibility, reliance on efficient and robust frameworks (such as Spring Boot for Java), and stringent adherence to security protocols. By addressing these factors during the design and development phases, teams can create a resilient and high-performing game application capable of meeting the demands of a diverse user base.

## [System Architecture View](#_ilbxbyevv6b6)

Logical Architecture

The system utilizes a multi-tier architecture, which includes the following layers:

Client Layer

Users interact with the game through client applications available on various platforms, including Windows, Mac, Linux, and mobile devices. Clients send requests (e.g., to join games, create teams, or add players) via HTTP APIs. This layer can be implemented using web browsers or native mobile/desktop applications.

Application Layer

The core logic of the system resides in the game service (GameService), which employs the Singleton pattern to manage game-related operations such as adding games, teams, and players. This layer enforces important rules, including unique names and object relationships (Game -> Team -> Player), by utilizing object-oriented principles.

Data Layer

Persistent storage for games, teams, and players is managed through a database (e.g., MySQL, PostgreSQL). The data layer communicates with the application layer via an ORM (Object-Relational Mapping) or through direct queries.

Physical Architecture

The system is deployed in a distributed environment, comprising the following components:

Web Server

This server hosts the application logic using a framework like Java Spring Boot. It exposes APIs for client-server communication.

Database Server

The database server is responsible for storing game data persistently. For scalability, a cloud database system (e.g., Amazon RDS, Google Firebase) is recommended.

Networking

Communication between components occurs over a secure network (HTTPS with TLS). Load balancers can distribute traffic across multiple application instances to enhance scalability.

Diagram Description

The architecture offers several benefits:

Modularity: There is a clear separation of concerns across the different layers.

Scalability: The application and database layers can be scaled independently.

Security: Data is encrypted both in transit and at rest.

Fault Tolerance: The distributed deployment ensures reliability in the event of server failure.

For future expansions, this architecture can integrate microservices to isolate functionalities such as game management, user authentication, and other features into separate services.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram for the game application includes the following components:

Entity Class

This is a base class that provides a unified structure for all entities, incorporating attributes such as `id` and `name`.

Game, Team, and Player Classes

These classes inherit from the Entity class and extend its functionality to manage relationships. For example, the Game class manages a list of Team objects, while the Team class manages a list of Player objects.

GameService

This component implements the Singleton pattern, ensuring a single point of control for all game-related operations. It facilitates the addition of games, teams, and players while enforcing name uniqueness through the Iterator pattern.

Object-Oriented Principles

- Encapsulation

Attributes are kept private, with access provided through public getters and setters.

- Inheritance

The Game, Team, and Player classes extend the Entity class.

- Polymorphism

Methods like `toString()` are overridden to deliver class-specific behavior.

This structure promotes efficient fulfillment of software requirements while ensuring modularity and scalability.

**"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** | This service is dependable for hosting, featuring built-in tools such as Apache and Docker; however, it comes at a higher cost. | Preferred for server hosting because of its performance, cost efficiency, and scalability. | The solution is flexible and widely used, but it may come with higher licensing costs for enterprise use. | The hosting capabilities are limited, making it suitable only for lightweight or serverless setups. |
| **Client Side** | |  | | --- | |  |  |  | | --- | | Development requires Mac systems, which leads to higher costs and longer setup times. | | Cost-effective and versatile, but may require additional tools for client testing. | It is the easiest to support because of its widespread usage and the availability of tools such as Visual Studio. | The design and testing must be lightweight and compatible with various screen sizes and operating system versions. |
| **Development Tools** | Tools such as Xcode and IntelliJ IDEA are frequently used. | Compatible with development tools such as Eclipse and IntelliJ IDEA. | Visual Studio, IntelliJ IDEA, and Eclipse are all fully compatible with each other. | The project requires SDKs for both iOS and Android. Specifically, you will need Xcode for iOS development and Android Studio for Android development. |

Comprehensive Comparative Review of Platforms

1. Does each operating platform have a server-based deployment method wherein the website will be hosted?

* Linux:

Yes, Linux is the most preferred platform for server hosting. It is widely used in enterprise and cloud environments. It supports robust web hosting solutions such as Apache, Nginx, and Tomcat. Linux is highly compatible with cloud platforms such as AWS, Google Cloud, and Microsoft Azure; it provides scalability and flexibility in server configurations. It also supports containerization example, Docker, and orchestration tools-for example, Kubernetes distributed application management.

* Mac:

Yes, macOS natively provides server features, such as Apache. However, macOS is uncommon for a high-level hosting environment because it does not have the same level of scalability and community support as Linux. For high-performance production servers, it is good for use in small-scale or development environments.

* Windows:

Yes, Windows Server supports IIS, which is robust for hosting .NET applications and other web services. Windows is widely used in enterprise settings due to its tight integration with Microsoft technologies and support for Active Directory and SQL Server. However, scalability may require significant investments in licensing.

* Mobile Devices:

No direct server-side deployment. Mobile devices were designed to be clients rather than servers. In the case of server-side functionality, serverless computing or hosting services like AWS Lambda and Firebase are good options.

1. What are the potential licensing costs for the server operating system?

* Linux:

Free and open-source versions of Linux, such as Ubuntu, CentOS, and Debian, do not have any licensing costs. Enterprise versions, such as Red Hat Enterprise Linux and SUSE, require a subscription for support and updates. These are also considerably cheaper than their commercial counterparts.

* mac:

macOS itself doesn't have any licensing fees, but hardware costs are expensive to run a Mac server. As an example, the cost of using Apple's Mac mini or Mac Pro as servers could be upwards of $700 for low-end and up to several thousand dollars for higher configurations.

* Windows:

Windows Server incurs licensing costs that vary based on the number of users and cores. For example, a Standard edition license costs around $500–$1000, while data center editions for larger-scale deployments are much more expensive. Additional costs for Client Access Licenses (CALs) can further increase expenses.

* Mobile Devices:

The costs of licensing are indirect and cloud-hosted services or serverless architectures. For example, AWS Lambda charges per compute time, while Firebase has tiered pricing depending on usage.

1. What does the application development process have to do to ensure compatibility with all web browser platforms and mobile devices?

* Responsive Design:

The application should be developed on responsive design principles to make sure it looks great on all screen sizes and resolutions. Using Bootstrap, Tailwind CSS, or Material-UI will be highly recommended to maintain consistency in user interface design.

* Cross-Browser Compatibility:

Extensive testing needs to be done to ensure compatibility with most web browsers, such as Chrome, Firefox, Safari, and Edge. One may use BrowserStack or Selenium for automated cross-browser testing.

* Progressive Web Application (PWA):

Developing the app as a PWA will ensure that it works offline, loads fast, and behaves natively on mobile devices. This approach uses modern web APIs to enhance functionality.

* Cross-Platform Mobile Frameworks:

Frameworks like React Native, Flutter, or Xamarin allow developers to write code once and deploy it on both iOS and Android platforms, reducing development time and costs.

* Back-End Compatibility:

The back-end APIs should be at least RESTful to ensure that communications between the client-side application and the server work seamlessly, regardless of the platform.

1. What are the effects of development tools on a development team? Consider whether multiple development teams may be needed.

* Unified Development Tools:

Tools such as IntelliJ IDEA, Visual Studio Code, and Eclipse provide a single environment for cross-platform application development. These tools lower the requirement for multiple teams since one codebase can be deployed on different platforms.

* Specialized Teams:

Even with unified tools, there might be certain platform-specific features, such as native UI/UX design for iOS or Android, that require dedicated expertise. Example:

* An iOS-specific team to follow Apple's Human Interface Guidelines.

An Android-specific team for handling variations in device sizes and Android versions.

* Collaboration:

Using cloud-based development environments, such as GitHub Codespaces and AWS Cloud9, allows teams to collaborate from different locations and maintain the same development practices.

1. Are there any licensing costs associated with the development tools?

* Linux:

Most of the tools, such as Eclipse and IntelliJ IDEA Community Edition, are free and open-source. The premium features of some tools, like IntelliJ IDEA Ultimate, require licensing, starting at $150–$500 annually.

* Mac:

Xcode, the primary development tool for iOS, is free but requires a Mac device. The Apple Developer Program, necessary for app distribution on the App Store, costs $99 annually.

* Windows:

Visual Studio Community Edition is free for small teams and individual developers. Larger teams or enterprises may need to purchase Professional or Enterprise editions, costing $500–$6000 annually depending on the version.

* Mobile Devices:

Android Studio does not cost anything for Android development. In the case of iOS, there is no cost in getting Xcode, but it requires device acquisition costs and costs regarding the Apple Developer Program.

## Recommendations

1. Operating Platform  
     
   It is recommended that Linux be the server-side operating platform simply because of its cost-effectiveness, reliability, and dominating factor in web-based distributed systems. Linux distributions such as Ubuntu Server or RHEL are quite suitable for scalable environments and have considerable community and enterprise support.

Scalability: Linux supports containerized deployment platforms (e.g., Docker) and orchestration systems (e.g., Kubernetes), allowing The Gaming Room to scale efficiently as user demand grows. These tools reduce server provisioning time and ensure fault tolerance.

Client-Side Compatibility: On the client side, it should ensure compatibility with Windows, macOS, iOS, and Android. In this regard, the adoption of cross-platform frameworks, such as React Native or Flutter, is highly beneficial because one codebase will be used to provide the same experience for all devices.

1. Operation System Architectures  
     
   The Linux kernel is a modular, general kernel-based architecture that is based on a preemptive multitasking capability with fine-grained resource control, especially applicable in the hosting of distributed applications. This includes features such as groups, allowing the granular division of running services between CPU, memory, and I/O resources for the sake of stable performance.

Client-Side Architecture: Windows and macOS are fitted with the ability to run most of the modern API calls and their runtime environments that may be used by a Linux-based server. Mobile platforms iOS and Android use lightweight app runtimes optimized for lightweight communication with a distributed system.

Middleware Integration: In this, implement middleware such as Node.js or Spring Boot to handle communications between varied client platforms and the Linux server.

1. Storage Management  
     
   Cloud-Based Storage: Leverage a scalable, distributed database system such as Amazon RDS for relational databases, like MySQL or PostgreSQL, or Firebase for NoSQL solutions. These databases provide features including but not limited to automatic scaling, high availability, and data encryption.

Data Synchronization: For a distributed environment, implement data replication across geographic locations for consistency. Tools like Amazon Aurora or MongoDB Atlas offer multi-region replication with low latency.

Backup and Recovery: Establish the development of automated backups every day, while at the same time following disaster recovery best practices. AWS Backup or Google Cloud's Data Backup could support seamless recovery solutions in case of data loss.

1. Memory Management  
     
   Java Virtual Machine: The Linux platform, combined with JVM-based languages such as Java or Kotlin, enables efficient memory management using garbage collection techniques to free up unused memory dynamically. This reduces the chances of memory leaks, which is crucial for long-running server-side applications.

Mobile Platform Optimization: Implement caching mechanisms, such as Redis for server-side and in-app caches for client-side\*\*, to minimize memory usage and improve real-time response times. On mobile platforms, utilize memory profiling tools such as Xcode Instruments for iOS and Android Profiler to optimize memory allocation.

Load Balancing of Memory Usage: Make use of Kubernetes Horizontal Pod Autoscaling to dynamically allocate memory and CPU to containers based on real-time application demand.

1. Distributed Systems and Networks  
     
   Communication Protocols: RESTful APIs are used for light and stateless communication between the server and client platforms. For more complex data interactions, consider GraphQL, which enables clients to request only the data they need, thus optimizing network use.

Containerization and Orchestration: Use Docker to package the software components into containers for consistent runtime environments across servers. Kubernetes can be used for automated deployment, scaling, and management of these containers in a distributed environment.

Load Balancing and Fault Tolerance: Utilize tools such as AWS Elastic Load Balancer or NGINX to distribute traffic across a pool of instances. Deploy in multiple availability zones to ensure minimal disruptions in service during an outage.

Dependency Management: Make use of utilities like Apache Kafka or RabbitMQ, which manage inter-component interaction in a system that is distributed. These message brokers ensure resiliency by queuing messages during temporary connectivity issues.

Offline Capability: This will allow mobile clients to cache data locally to enable limited functionality in case of network downtime and sync their data when the network comes back.

1. Security  
     
   Data in Transit: Make all communications between client and server over HTTPS with TLS 1.2 or 1.3 encryption to provide integrity against interception and tampering. Further, implement HSTS (HTTP Strict Transport Security) to enforce HTTPS.

Authentication of User: Use modern and secure authentication mechanisms such as OAuth 2.0 or JWT in an application. It shall help prevent unauthorized access; this framework will defend against very general types of attacks like hijacking or fixation of any active session.

Data in Storage Security: The critical data at rest must be encrypted by following standards, i.e., AES - 256. Data Base: Allow only RBAC so that access to important information is restricted.

Threat Monitoring: Utilize real-time monitoring tools such as AWS CloudTrail, Azure Monitor, or Google Cloud Security Command Center to identify and respond to security incidents. Platform-

Specific Protection: On mobile platforms, ensure secure coding practices are in place and sandboxing to isolate the app processes. Apply biometric authentication where appropriate, such as Touch ID or Face ID, to enhance user security.

These recommendations provide a robust framework for "Draw It or Lose It" by combining a robust Linux-based architecture with scalable storage, efficient memory management, and secure communication protocols. This approach supports the goals of The Gaming Room for expanding the game across multiple platforms while ensuring security, performance, and scalability in a distributed environment.