

**Draw It or Lose It**

Alexander Falatine

# CS 230 Project Software

# Dr. Kevin Eaton

# 10//202419

## 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 | 09/22/2024 | Alexander Falatine |  |

**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 wants to transition their current Android game Draw It or Lose It into a web-based game that is accessible across multiple platforms. The game is inspired by the 1980s television show Win, Lose or Draw, where teams guess what is being drawn. In the current Android version, a large library of stock images provides clues for teams to guess the puzzle. The new web-based version must keep the core gameplay elements while providing cross-platform compatibility, making sure that users can engage with the game from various devices such as desktop, mobile, and tablet browsers.

My solution focuses on developing a scalable web application using a distributed environment to ensure performance and reliability. By utilizing modern web technologies and cloud infrastructure, the game can handle multiple sessions simultaneously as well as multiple teams and players. The critical challenge is maintaining a single instance of the game in memory, ensuring the uniqueness of game and team names as well as smoothly transitioning the existing game mechanics to a distributed architecture.

## Requirements

The proposed solution will require implementing Singleton and Iterator design patterns to manage the lifecycle of the game, teams, and players. A central server to manage game state and player interactions across multiple sessions will also be implemented. Cross-platform compatibility must be used to allow seamless gameplay on desktop and mobile devices. Creating scalable infrastructure that can handle fluctuating user traffic with minimal latency will also be implemented. This approach ensures that The Gaming Room’s web-based game will be optimized for performance, user experience, and future scalability.

## [Design Constraints](#_2et92p0)

Cross-Platform Compatibility: The game must work seamlessly across multiple platforms, including Windows, macOS, Android, and iOS, via a web browser. This constraint implies using web standards like HTML5, CSS3, and JavaScript frameworks (e.g., React or Angular) to ensure that the application performs consistently regardless of the operating system or browser. However, optimizing various screen sizes and input methods (mouse, keyboard, touchscreen) will require careful design and testing, increasing development time and complexity.

Scalability and Performance: The web-based game will need to handle multiple users simultaneously, with the potential for increased traffic during peak usage times. This requires scalable cloud infrastructure and efficient backend architecture, likely using a framework such as Node.js for real-time communication.

Singleton Game Instance: Only one instance of the game should exist in memory at any given time. This constraint necessitates implementing the Singleton design pattern to ensure that only one game is live per server or session.

Unique Identifiers for Game, Teams, and Players: All game, team, and player names must be unique to avoid conflicts. This will require a centralized system for name validation in real time, ensuring that names are checked against existing entries in the database.

Web-Based Distributed Environment: The game will operate in a distributed environment, where client and server communication are crucial for real-time gameplay. This means handling issues like latency, network reliability, and session persistence. Additionally, distributed environments can introduce synchronization challenges, where game state must be kept consistent across all connected clients, leading to increased complexity in managing data consistency and preventing race conditions.

## [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 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.

The UML diagram is a process diagram of The Gaming Room that illustrates the structure and relationships between various classes designed to implement this web-based game. To start, the ProgramDriver contains the main method which suggests it is the entry point of the application of which it runs from. There is a class connected called SingletonTester of which ensures that the Singleton pattern is correctly running on the main method.

The Entity class here represents a base class for objects that need a unique identifier (ID) and a name which it gives to three classes that inherit from it, the Game, Team, and Players classes. The Team class contains a list of players to manage what players are on what team and has methods to add players and represent team data as a string. The Player class represents individual players with player-specific details. The Game class Manages multiple teams through a <list>Team method and includes methods to add teams to the game and convert game details into a string format.

The final class I see is a GameService Class which does not inherit from Entity as most of the program does. It does manage all game instances via the List<Game> method and handles unique ID tracking for games, teams, and players with nextGameId, nextPlayerId, and nextTeamId methods respectively.

**The object-oriented principles at play here are as follows:**

Inheritance: Demonstrated by Player, Team, and Game classes inheriting from the Entity class. This promotes code reusability and a hierarchical structure where common attributes and methods are defined in the Entity class.

Encapsulation: The use of private attributes and public methods in all classes ensures that internal states of objects are hidden from external manipulation, seen in all classes with private data and public getters or utility methods.

Association: Game class contains a list of Team objects, and Team contains a list of Player objects, representing a “one-to-many" relationship.

Aggregation: Demonstrated by the relationship between GameService and Game, where GameService has a list of games, but their lifecycles are independent from each other.

## [Evaluation:](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Pro:  -Known for robust performance and security.  - Straightforward, secure, and high-performance environment.  -Built-in support for server applications through Apache or Nginx.  - Integrate well with macOS server environments.  Con:  -Higher costs/less flexible hardware configurations  - Server use limited  - Licensing costs for servers can be significantly higher than Linux. | Pro:  -Operations stability, security, and flexibility highly favorable.  - supports a wide range of hardware and is extensively customizable  -Supports wide range of hardware/ customizability  -Scalable - Ideal platform for server-side hosting due to wide support for server tools like Apache, Nginx, and MySQL.  Con:  -Steep learning curve requiring expert knowledge | Pro:  -Intuitive user interface  - Strong compatibility with other software solutions.  - Integrates well with other Microsoft products and services.  - Provides reliable performance with IIS support.  - Comprehensive features for hosting.  Con:  -Costly due to licensing fees  - Requires more resources for operation.  -Less customizable than Linux  - Can incur additional licensing costs. | Pro:  -lightweight  -Portable  -Built-in connectivity - Mobile devices can connect to backend web applications using RESTful APIs, though challenges with bandwidth and latency may arise.  Con:  limited processing power, memory, and storage comparatively. |
| **Client Side** | -Safari provides strong HTML5 support, but developers may need to ensure compatibility with platform-specific optimizations and accessibility features.  - Consideration of ensuring compatibility with Safari is required.  - Possibly expensive if native integration or optimizations are needed.  - Developers need to be familiar with macOS-specific standards and accessibility features. | -Ensuring compatibility across different Linux distributions and browsers can be a challenge, requiring thorough testing and potential customizations.  - Developing for Linux clients can be less straightforward due to the diversity of Linux distributions and desktop environments.  - May require extensive testing to ensure consistent user experiences.  - Web browsers like Firefox and Chromium are commonly used and support modern web standards. | -Windows supports a variety of browsers, making client-side development relatively straightforward.  - A widely used OS, and developing client-side for Windows involves ensuring compatibility with popular browsers like Chrome and Edge.  - Costs are moderate, but the large user base of Windows means testing across different versions and system configurations can be time-consuming. | -Mobile apps need to be responsive, and expertise in UI/UX design is essential for creating seamless user experiences across devices.  - Developing mobile devices means ensuring compatibility with Android and iOS.  - Can involve significant costs in terms of development and testing due to mobile apps requiring frequent updates and optimizations due to diverse hardware.  - Expertise in responsive design tools such as UI/UX required. |
| **Development Tools** | Tools commonly used for developing on Mac include Xcode for native applications and web development tools like Visual Studio Code, Atom, and Sublime Text for web-based applications.  Swift for native apps and JavaScript, Python, or Ruby for web applications are more popular here.  Developers may need to be cross-trained to manage both Swift-based native apps and JavaScript-based web applications. | Linux developers have access to a vast array of programming tools and environments, including GCC for C/C++ development, Python, Java, and JavaScript for backend technologies.  Web technologies such as Apache, Nginx, Node.js, and database systems like MySQL and PostgreSQL are commonly used in Linux environments. Licensing costs for development tools are minimal on Linux, but expertise in multiple programming environments is necessary. | Development on Windows usually involves tools like Visual Studio for .NET applications or application creation. Visual Studio Code for web development is common. And languages such as C#, C++, Python, VB.NET, and ASP.NET for server-side scripting as well as other types of application development.  Windows-based teams might require licensing for Visual Studio and other tools, increasing costs. | Development tools for mobile devices include Android Studio for Android apps and Xcode for iOS apps.  Cross-platform frameworks like React Native, Xamarin, and Flutter can reduce development time, though expertise in both Android and iOS environments is necessary.  This can be cross-platformed by using tools like React Native, Xamarin, and Flutter from what I've seen, though I am less familiar with mobile coding. These tools support languages like Kotlin, Java (Android), Swift, and Objective-C (iOS), as well as JavaScript. |

## 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 expanding *Draw It or Lose It* across multiple computing environments, I recommend using **Linux** as the primary server platform. Linux offers robust stability, flexibility, and scalability. These are critical aspects needed to run a distributed web-based game. It is suited for handling the backend infrastructure due to its strong support for server tools such as Apache, Nginx, Node.js, and MySQL. Also, Linux is highly customizable and the fact there is a lack of licensing fees lowers overall costs and maintenance of the program, making it not just the technological choice but also the economical choice as well.
2. **Operating Systems Architectures**: Linux offers a modular architecture, which means it's made up of individual components that can be loaded and unloaded dynamically. This type of architecture creates an environment where only the required components are installed and running, reducing the potential for bloat and enhancing performance simultaneously. Its multi-user and multi-tasking capabilities allow the operating system to manage multiple connections and services at once. This will allow new features to be added to the modules while the system is running. Its components can be updated or replaced without affecting the entire system as well, all of which helps in supporting multiple sessions and teams within the game as it is running. The system's “monolithic kernel architecture” (The “kernel” consists of a single, large executable that includes various services such as memory management, file system management, device drivers, and process management) offers superior I/O performance, making sure that communication between hardware components and processes is efficient and safe, both of which are important factors for real-time gameplay on a web-based platform.
3. **Storage Management**: Linux provides powerful storage management options, including logical volume management (LVM) and RAID configurations, ensuring both flexibility and redundancy. For the game’s backend memory, a MySQL database will store player data, game states, and team information. MySQL’s compatibility with Linux and support for ACID (Atomicity, Consistency, Isolation, Durability) properties will ensure data integrity and consistency. Additionally, cloud-based storage options such as Amazon S3 or Google Cloud Storage can be integrated with Linux to provide scalability and reliable data backups.
4. **Memory Management**: Linux utilizes advanced memory management techniques such as virtual memory, demand paging, and swapping. This makes sure that memory usage is optimized even during peak usage. It uses a “least recently used” (LRU) algorithm to manage cache memory effectively where the game’s processes and sessions in memory are prioritized correctly. This helps prevent bottlenecks during high user activity, which means the game remains responsive and maintains performance across multiple sessions.
5. **Distributed Systems and Networks**: The Draw It or Lose It game will rely on a distributed system to enable real-time communication between clients and servers. Linux, having a strong support for microservices architectures and Docker containers (a set of platforms as a service product that use OS-level virtualization to deliver software in packages called containers), will allow the game components to run efficiently across multiple servers. Nginx or HAProxy can be utilized for load balancing which will make sure that no single server becomes overwhelmed during high traffic periods. Network dependencies will be mitigated using message queues (A buffer that temporarily stores messages, and endpoints that allow software components to connect to the queue) such as RabbitMQ or Kafka to ensure game state synchronization, even during brief connectivity disruptions. This distributed model will ensure that all devices stay in sync and gameplay continues smoothly despite potential network fluctuations.
6. **Security**: Linux is known for its strong security capabilities, including built-in tools like SELinux (Security-Enhanced Linux) for mandatory access controls to the program. To protect user data across platforms, the platform will employ encryption protocols such as TLS (Transport Layer Security) for secure communication between clients and servers. The use of firewalls (like iptables) and intrusion detection systems (such as Fail2ban for example) will enhance network security even more. Also, user authentication will rely on OAuth 2.0 or Basic Authentication with encrypted passwords to enforce safe logins and protect user data as well. Security monitoring tools and regular updates must also be implemented as the game expands and maintaining these protective measures with said updates will further protect the system from vulnerabilities