

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.3

## Table of Contents

[CS 230 Project Software Design 1](#_Toc923313610)

[Table of Contents 1](#_Toc477283398)

[Document Revision History 2](#_Toc85568553)

[Executive Summary 2](#_Toc300319566)

[Requirements 3](#_Toc56078877)

[Design Constraints 3](#_Toc1684744758)

[System Architecture View 3](#_Toc1733250533)

[Domain Model 3](#_Toc174708797)

[Evaluation 4](#_Toc2018793947)

[Recommendations 4](#_Toc1502273980)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/15/2023 | Michael Rosenbaum | Provided the Executive Summary, Requirements, and Design Contraints. |
| 1.1 | 07/16/23 | Michael Rosenbaum | Added System Architecture View, Domain Model, Evaluation, and Recommendations. |
| 1.2 | 7/30/23 | Michael Rosenbaum | Double checked Evaluation section. |
| 1.3 | 8/13/23 | Michael Rosenbaum | Double checked Recommendations section. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has approached Creative Technology Solutions (CTS) with the objective of developing a web-based version of their existing game, "Draw It or Lose It," which is currently available as an Android app.

## Requirements

Business Requirements:

* Cross-platform Compatibility: The web-based game application must be accessible and playable on multiple platforms, including desktop computers, tablets, and mobile devices, to reach a broader audience and enhance user engagement.
* Multiplayer Interaction: The game should facilitate real-time multiplayer interaction, enabling multiple teams with multiple players to participate and compete simultaneously, fostering a competitive and collaborative gaming experience.
* Unique Name Management: The application must implement a system to ensure that game and team names are unique, allowing users to verify name availability and prevent any potential confusion or overlapping instances.

Technical Requirements:

* Database Integration: The game application requires seamless integration with a database system to manage user profiles, game progress, and unique name checks for games and teams.
* Singleton Design Pattern: Implement the Singleton pattern to enforce that only one instance of the game exists in memory at any given time, eliminating conflicts between concurrent game sessions and ensuring a smooth gameplay experience.
* Stock Drawing Library: Develop a robust library of stock drawings that the application can render as clues, ensuring a diverse range of puzzles for players to guess during each round of the game.

## [Design Constraints](#_2et92p0)

Design Constraints for Developing the Game Application in a Web-based Distributed Environment:

* Network Latency: In a distributed web environment, network latency can be a significant constraint. As the game involves real-time multiplayer interaction, any delays in data transmission and communication between players and the game server can impact the gameplay experience.
* Scalability: The application needs to handle a potentially large number of concurrent users playing across multiple teams and games. Ensuring scalability is essential to accommodate varying levels of user traffic without compromising performance.
* Security: As a multiplayer game, the application must address security concerns to protect user data, prevent unauthorized access, and safeguard against potential cheating or hacking attempts.
* Cross-browser Compatibility: The game must be compatible with various web browsers, as users may access the application from different devices and platforms.
* Synchronization: Maintaining synchronization between multiple players in real-time is crucial for a seamless and fair gaming experience.
* Bandwidth Constraints: Users with limited internet bandwidth should be able to play the game without significant performance degradation, necessitating efficient data handling and minimal data transfer.

Implications on Application Development:

* Optimized Network Communication: To mitigate the effects of network latency, the application's communication protocol should be optimized for real-time data exchange, minimizing the time lag between player actions and responses from the server.
* Distributed Architecture: Employ a distributed architecture that allows the game to scale horizontally to handle increased user traffic, ensuring that the system can accommodate a growing player base without compromising performance.
* Encryption and Authentication: Implement robust encryption and authentication mechanisms to secure user data and prevent unauthorized access to the game's backend infrastructure.
* Compatibility Testing: Conduct extensive cross-browser compatibility testing to ensure that the game functions correctly and consistently across various web browsers and devices.
* Real-time Synchronization: Utilize techniques like WebSockets or long-polling to achieve real-time synchronization between players, enabling seamless collaboration and competitive gameplay experiences.
* Data Optimization: Optimize data transmission to minimize bandwidth usage, compressing and transmitting only essential game-related information to reduce the strain on users with limited internet connectivity.

## [System Architecture View](#_ilbxbyevv6b6)

Subsystem Architecture:

* User Management Subsystem: Responsible for handling user registration, authentication, and profile management. It interfaces with the database to store user information securely.
* Game Logic Subsystem: Manages the game rules, puzzle generation, and real-time gameplay interactions, ensuring fair and synchronized gameplay for multiple teams.
* Stock Drawing Library Subsystem: Stores and manages the library of stock drawings, providing the game logic subsystem with a variety of puzzles to render during gameplay.

Physical Components/Tiers:

* Presentation Tier: The client-side tier comprises web browsers on various devices that render the game interface and allow players to interact with the game.
* Application Tier: The application server tier hosts the game logic subsystem, handling real-time interactions, user authentication, and game state management.
* Data Tier: The database server tier stores user profiles, game progress, and stock drawings, facilitating data retrieval and updates for the application and game logic subsystems.

## [Domain Model](#_8h2ehzxfam4o)

The Entity class is the super class while the Game, Player, and Team class inherit from the it. The

GameService class references the Game class, the Game class has a reference to the Team class, and

the Team class has a reference to the Player class. The ProgramDriver class is where any executions take

place in the program, it uses the SingletonTester class. The SingletonTester class uses the

ProgramTester class which allows one game to go on at a time with multiple teams and players from a specific team list and player list.

**"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** | Provides a Unix-based operating system (macOS), making it well-suited for web application development due to its Unix-like environment, which shares similarities with Linux. Advantages: Seamless integration with Apple's development tools and frameworks, such as Xcode and Swift, provides a robust and efficient development environment for building web-based software. Weaknesses: Limited market share compared to Windows, potentially reducing the reach of the web application to a smaller user base. | Open-source operating system with a wide range of distributions tailored to specific needs, including server-focused distributions like Ubuntu Server and CentOS. Advantages: High stability, robust security, and extensive support for a variety of programming languages and frameworks make Linux an excellent choice for hosting and developing web-based software applications. Weaknesses: May require more technical expertise to set up and maintain compared to Mac or Windows, as certain configurations and software installations might be more manual in nature. | A widely used operating system with a substantial user base, making it suitable for targeting a broad audience. Advantages: Strong development tool support, including Visual Studio and .NET frameworks, provides a familiar and feature-rich environment for building web applications on Windows. Weaknesses: Historically, Windows has been less preferred for server-side hosting compared to Linux due to potential security concerns and higher resource consumption. | Smartphones and tablets, run on various operating systems, including iOS (for Apple devices) and Android (for a wide range of devices). Advantages: Mobile devices allow the application to reach a vast and diverse user base, providing excellent potential for user engagement and accessibility. Weaknesses: Development considerations for different operating systems, screen sizes, and device capabilities can increase development complexity and time. |
| **Client Side** | MacOS app development using Swift or web-based technologies such as HTML, CSS, and JavaScript. Development costs may be higher if targeting only Mac users due to the limited market share. | Web-based applications using standard web technologies like HTML, CSS, and JavaScript, making it relatively platform-independent. Development costs may be moderate, but expertise in Linux-based hosting and deployment is essential. | Web-based applications with technologies like HTML, CSS, and JavaScript. The development cost might be similar to Linux, but expertise in Windows hosting environments would be required. | Mobile apps for both iOS and Android using Swift or Objective-C for iOS and Java or Kotlin for Android. Development costs may be higher due to the need for separate app development for each platform. |
| **Development Tools** | Xcode is the primary Integrated Development Environment (IDE) for Mac app development, using Swift as the programming language. Web-based software can be developed using standard web development tools like Visual Studio Code or Sublime Text. | Supports a wide range of programming languages like Python, Node.js, Ruby, PHP, etc., and developers can use popular IDEs such as Visual Studio Code, Sublime Text, or Vim for web-based software development. | Visual Studio is the primary IDE for Windows app development using .NET languages like C# or VB.NET. Web-based software development can be done using Visual Studio Code or other web development IDEs. | iOS app development requires Xcode and Swift/Objective-C, while Android app development utilizes Android Studio and Java/Kotlin. Web-based software can be developed using standard web development tools as mentioned earlier. |

## Recommendations

1. **Operating Platform**: Given the client's desire to expand Draw It or Lose It to other computing environments and support multiple platforms, I recommend using a web-based application architecture with Linux as the operating platform. Linux offers excellent scalability, security, and cross-platform compatibility, making it an ideal choice for hosting and deploying web-based game applications.
2. **Operating Systems Architectures**: Linux is an open-source operating system built on a Unix-based architecture. It provides a stable and secure environment, making it suitable for hosting web applications across different platforms. Its modular design allows for easy customization and seamless integration with various components.
3. **Storage Management**: A relational database management system (RDBMS) like PostgreSQL or MySQL can be utilized on the Linux server. These database systems offer robust data handling capabilities, ensuring the secure storage and retrieval of user profiles, game progress, and stock drawings.
4. **Memory Management**: Linux's memory management is highly efficient, utilizing techniques like virtual memory, paging, and memory allocation algorithms to optimize the use of system resources. This ensures that the Draw It or Lose It software running on the recommended operating platform performs well and is responsive to user actions.
5. **Distributed Systems and Networks**: To achieve communication between various platforms and enable a distributed gaming experience, the web-based application architecture on Linux can employ WebSocket technology. WebSockets provide real-time bidirectional communication between clients and the server, allowing seamless interaction between players on different devices. Dependencies between components within the distributed systems and networks must be carefully considered. Redundancy and failover mechanisms should be implemented to handle network outages and ensure continuous gameplay without significant interruptions. Additionally, load balancing techniques can be used to distribute incoming requests evenly across multiple servers, improving the application's overall performance and reliability.
6. **Security**: Security is a top priority for the client, and Linux provides robust security capabilities. The recommended operating platform can implement secure communication using HTTPS/SSL encryption to protect user information during data transmission. Linux's built-in firewall and access control mechanisms help safeguard the server from unauthorized access and potential cyber threats. The web-based application architecture can enforce secure user authentication and authorization procedures to ensure that only authenticated users have access to the game and their respective data. Proper encryption techniques can be employed to protect sensitive user data stored in the database, mitigating the risk of data breaches. A web-based application architecture on Linux serves as a well-suited and reliable solution for The Gaming Room's expansion of Draw It or Lose It to various computing environments. The combination of Linux's stability, scalability, and security features, along with web technologies like WebSockets and secure communication protocols, will enable the seamless communication and secure user experience desired by the client.