

Draw It or Lose It Expansion

# **CS 230 Project Software Design Document**

Version 1.0

## Table of Contents

Contents

[**CS 230 Project Software Design Document** 1](#_Toc180107376)

[Table of Contents 2](#_Toc180107377)

[Document Revision History 2](#_Toc180107378)

[Executive Summary 3](#_Toc180107379)

[Requirements 3](#_Toc180107380)

[Design Constraints 3](#_Toc180107381)

[System Architecture View 3](#_Toc180107382)

[Domain Model 3](#_Toc180107383)

[Evaluation 4](#_Toc180107384)

[Recommendations 6](#_Toc180107385)

[Operating System Architectures 6](#_Toc180107386)

[Storage Management 6](#_Toc180107387)

[Memory Management 6](#_Toc180107388)

[Distributed Systems and Networks 7](#_Toc180107389)

[Security 7](#_Toc180107390)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/18/2024 | Cassim Mabirizi | Initial software design document |
| 2.0 | 10/03/2024 | Cassim Mabirizi | Final document, includes evaluation of platforms for hosting and client support |

## [Executive Summary](#_sbfa50wo7nsh)

The **"Draw It or Lose It"** application is a game that allows players to compete in drawing-based challenges. Initially available only on Android, the application is now being expanded into a web-based platform, enabling it to run on multiple operating systems, including Linux, Mac, and Windows, as well as mobile platforms (iOS and Android).

This document evaluates the suitability of different operating platforms for hosting the game and supporting its client-side requirements. It also analyses the development tools and languages needed to build the game for deployment on multiple platforms, while considering factors such as licensing costs, scalability, and development time.

## Requirements

The client, **The Gaming Room**, needs to expand the "Draw It or Lose It" game to support desktop platforms (Linux, Mac, Windows) and mobile devices (iOS, Android). The application will be accessible through modern web browsers and must provide a responsive user experience across devices.

**Key requirements**:

1. Support thousands of concurrent users via a scalable server-based solution.
2. Deliver a responsive HTML interface to multiple types of clients (desktop and mobile).
3. Ensure compatibility with web browsers and mobile devices.
4. Implement proper server-side hosting for scalability and multi-threading.

## [Design Constraints](#_2et92p0)

To host a web-based, distributed game application, several constraints must be addressed:

* **Scalability**: The server infrastructure must support scaling as the number of players grows, ensuring performance is not impacted.
* **Compatibility**: The application must be compatible across platforms and devices while maintaining the same user experience.
* **Cost**: Development tools and server infrastructure must be cost-effective, considering both licensing and operational expenses.
* **Security**: As the game expands to a larger user base, strong encryption and authentication measures must be implemented to protect player data and prevent unauthorized access.

## [System Architecture View](#_ilbxbyevv6b6)

There are no specific requirements for the system architecture view in this project, but the design will follow a distributed web-based model. The server-side will handle game state management, user authentication, and concurrent gameplay, while the client-side (running in web browsers or mobile apps) will render the game interface and communicate with the backend using RESTful APIs.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram illustrates the relationships between major classes such as **GameService**, **Game**, **Player**, and **Team**. The **GameService** class implements the Singleton design pattern to ensure only one instance manages all active games. Each game contains multiple teams, and each team consists of several players. Object-oriented programming (OOP) principles, such as encapsulation and composition, are employed to ensure scalability and performance as more games, teams, and players are added.

**"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)

This section evaluates each operating platform (Linux, Mac, Windows, iOS, Android) for server-side hosting and client-side compatibility.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac OS supports server-based deployment but may require additional licensing for tools like macOS Server. The server platform is stable but generally not as common for large-scale deployments. | Linux is the most commonly used OS for web hosting due to its open-source nature and high scalability. Most cloud providers offer Linux hosting services, making it cost-effective. | Windows offers Windows Server, which is highly reliable but comes with significant licensing costs. It also has built-in tools like IIS for web hosting. | Mobile devices do not directly host servers but can interact with web-based applications via browsers or native apps that communicate with backend servers. |
| **Client Side** | Requires web browser compatibility testing, but macOS generally supports modern web standards. Additional time may be required for optimizing performance on Safari. | Linux desktop environments support modern web browsers like Chrome and Firefox, ensuring compatibility with the web-based interface. | Windows offers extensive support for web browsers and will require testing for compatibility with Edge, Chrome, and Firefox. | Developing for iOS and Android will require building responsive designs and ensuring compatibility across different screen sizes and performance variations. |
| **Development Tools** | Xcode can be used to develop and test applications for Mac, with licensing costs for deployment on App Store or distribution tools. | Linux allows for a wide variety of open-source tools like Eclipse, IntelliJ, and command-line utilities that are cost-effective for developers. | Visual Studio is commonly used for Windows-based development, but it comes with licensing costs. Windows also supports many other IDEs like JetBrains. | Android Studio and Xcode will be required for mobile development, both of which have steep learning curves and potential licensing costs for app distribution. |

## Recommendations

**Operating Platform**

For expanding "Draw It or Lose It" to various platforms, I recommend using a **Linux-based server platform** for the server-side operations. Linux is well-suited for hosting web-based applications due to its flexibility, scalability, open-source nature, and strong community support. It supports a wide range of programming languages and development tools, making it suitable for multi-platform deployment. Furthermore, Linux offers robust performance and efficient resource management, which is crucial for scaling applications to support more users.

### Operating System Architectures

The selected architecture for the Linux operating platform will be a **64-bit architecture**, which is common in modern server environments. The 64-bit architecture allows for larger memory address space, improved performance for computationally intensive tasks, and better handling of concurrent requests. This is beneficial for hosting a web-based game server where many simultaneous connections may occur.

**Modular Design**: Linux operating systems use a modular design, where the kernel can load and unload modules dynamically. This enables the system to run efficiently with minimal resource consumption and allows custom configuration tailored for the game server.

**Kernel-Level Memory Management**: Linux provides robust memory management capabilities at the kernel level, including virtual memory management, which improves the efficiency of multi-threaded applications.

**POSIX Compliance**: This ensures compatibility with different platforms and enables easier integration with other systems.

### Storage Management

For storage management, I recommend using a **relational database management system (RDBMS)**, such as **MySQL** or **PostgreSQL**, in conjunction with a **file-based storage system** for handling game data, user profiles, and media assets.

**MySQL/PostgreSQL**: These databases provide high performance, data integrity, and are capable of handling large volumes of transactions. They also support complex queries, which will be useful for the game logic, user stats, and leaderboards.

**File System Storage (e.g., Ext4 or Btrfs)**: For Linux, using a file system like Ext4 or Btrfs offers reliability and supports large file sizes, which is beneficial for storing assets such as images and audio clips used in the game.

### Memory Management

Linux employs various **memory management techniques** that are advantageous for hosting "Draw It or Lose It."

**Virtual Memory**: This technique allows processes to use more memory than physically available by utilizing disk space. It helps manage memory allocation effectively when multiple users are playing the game simultaneously.

**Paging and Swapping**: The system can swap inactive processes out of physical memory to free up space for active processes, ensuring optimal performance even under heavy load.

**Shared Memory**: This technique enables processes to share common memory segments, which can be useful for caching frequently accessed game data.

### Distributed Systems and Networks

To enable "Draw It or Lose It" to operate seamlessly across multiple platforms, a **distributed system architecture** using **RESTful APIs** for communication is recommended. The game can be hosted on a centralized server, while clients (users’ devices) can communicate with the server over the internet.

**Microservices Architecture**: Using a microservices approach allows the game services (e.g., game state management, user authentication, and game logic) to be modularized. Each service can run independently, making it easier to scale and maintain.

**Network Communication**: Utilize **WebSockets** for real-time communication between the server and clients. This allows for real-time game updates and user interactions.

**Load Balancing and Fault Tolerance**: Implement load balancing to distribute the traffic across multiple servers and use redundancy to handle outages or server failures.

### Security

Security is a critical consideration for protecting user data and ensuring safe gameplay. The following measures can be taken to secure the platform:

**User Authentication and Authorization**: Implement multi-factor authentication (MFA) and role-based access control (RBAC) to restrict access to the game services. Secure password hashing techniques like bcrypt should be used.

**Data Encryption**: Use **TLS/SSL** to encrypt all communications between the server and clients. Data at rest should be encrypted using file system encryption methods or database encryption features.

**Firewall and Intrusion Detection**: Set up firewalls to filter out unauthorized access and intrusion detection systems (IDS) to monitor suspicious activities.

**Secure APIs**: Implement API rate limiting and validation checks to prevent abuse or attacks like SQL injection or cross-site scripting (XSS).