

# A blue and black text Description automatically generated

# **CS 230 Project Software Design Template**

**Version 3.0**

## Table of Contents

[A blue and black text

Description automatically generated 1](#_Toc172020678)

[**CS 230 Project Software Design Template** 1](#_Toc172020679)

[Table of Contents 2](#_Toc172020680)

[Document Revision History 2](#_Toc172020681)

[Executive Summary 3](#_Toc172020682)

[Requirements 3](#_Toc172020683)

[Design Constraints 3](#_Toc172020684)

[System Architecture View 4](#_Toc172020685) - 6

[Domain Model 7](#_Toc172020686)

[Evaluation 8](#_Toc172020687)

Recommendations 9-12

## [Document Revision History](#_grjogdjh5fi8)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 3.0 | 07/16/24 | Joshua Hale | Project 3 submission |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room's current game application, Draw It or Lose It, is an Android-based game like the 1980s TV show Win, Lose or Draw. To expand its reach, the client aims to make the game accessible across multiple platforms (Linux, Mac, Windows, iOS, Android). This document evaluates these platforms in terms of server-side hosting capabilities, client-side development considerations, and relevant development tools. The goal is to provide a comprehensive analysis that will guide The Gaming Room in making an informed decision for expanding their application to a distributed web-based environment.

## Requirements

The client requires the game application to be web-based, allowing it to scale for thousands of players, and accessible on multiple platforms. The solution must include considerations for licensing costs, development tools, time, and expertise required for creating a modern, responsive HTML interface compatible with all web browsers and mobile devices.

## [Design Constraints](#_2et92p0)

Developing a web-based game application involves certain constraints, such as ensuring cross-platform compatibility, scalability, and maintaining a responsive user interface. Additionally, the choice of development tools, licensing costs, and the expertise required for each platform will influence the overall development process.

## [System Architecture View](#_ilbxbyevv6b6)

The system architecture for "Draw It or Lose It" consists of several key components that interact with each other to deliver the game experience to users. These components are divided into client-side and server-side elements. Below is an overview of the architecture:

**Client-Side Components:**

1. **Web Browser (Desktop) / Mobile App:**
   * **Description:** The user interface of the game that players interact with. The web version runs in a browser, while the mobile version runs as a native app.
   * **Technologies:** HTML, CSS, JavaScript, React (or similar framework), Swift/Objective-C for iOS, Java/Kotlin for Android.
2. **HTTP Client:**
   * **Description:** Manages communication between the client-side application and the server.
   * **Technologies:** Axios, Fetch API.

**Server-Side Components:**

1. **Web Server:**
   * **Description:** Hosts the web application and handles incoming HTTP requests from clients.
   * **Technologies:** Apache, Nginx.
2. **Application Server:**
   * **Description:** Contains the game logic and handles game-related operations such as managing game states, player interactions, and scoring.
   * **Technologies:** Node.js, Express.js (or similar backend framework).
3. **Database:**
   * **Description:** Stores persistent data including user information, game states, and scoring records.
   * **Technologies:** MySQL, MongoDB.
4. **API Gateway:**
   * **Description:** Serves as the single entry point for all client requests, routing them to the appropriate backend services.
   * **Technologies:** API Gateway, Kong.

**Communication:**

* **RESTful API:**
  + **Description:** Defines the endpoints for client-server communication, ensuring that the client can request data and the server can respond with the appropriate information.
  + **Technologies:** JSON over HTTP, REST principles.

**Security:**

* **Authentication Service:**
  + **Description:** Manages user authentication, ensuring that only authorized users can access the game.
  + **Technologies:** OAuth 2.0, JWT (JSON Web Tokens).

**Diagram:**

A diagram of a software application

Description automatically generated

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram provided demonstrates object-oriented programming principles such as encapsulation, inheritance, and polymorphism. These principles ensure the software's efficiency in fulfilling the game's requirements, such as rendering images, managing game rounds, and facilitating user interactions.

**"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** | Mac offers a robust environment for hosting web applications, leveraging Unix-based stability and scalability. However, licensing costs for macOS servers can be high. | Linux is renowned for its stability, security, and cost-effectiveness (often free). It supports various server environments like Apache and Nginx, making it ideal for scalable web applications. | Windows Server provides strong support for enterprise applications with integrated solutions like IIS. However, licensing can be costly. | Mobile devices primarily act as clients rather than hosts. However, Android and iOS can run local servers for development purposes. |
| **Client Side** | Developing for Mac requires consideration of its unique user interface standards. Costs involve acquiring macOS-specific development tools and expertise. | Linux development can be cost-effective due to open-source tools. However, it may require specialized knowledge for seamless integration. | Windows development is supported by a wide range of tools, such as Visual Studio. Costs involve licensing and the need for expertise in Windows-specific technologies. | Developing for mobile devices involves using tools like Xcode for iOS and Android Studio for Android. Costs include licensing fees for tools and platforms, and the expertise required for each. |
| **Development Tools** | Development on Mac uses tools like Xcode and languages such as Swift and Objective-C. Licensing costs for these tools are generally included with macOS. | Linux development leverages tools like GCC, Python, and open-source IDEs like Eclipse. Costs are minimal, but expertise in Unix-based systems is essential. | Windows development uses tools like Visual Studio, supporting languages like C# and .NET. Licensing for these tools can be expensive. | Mobile development uses platform-specific tools like Android Studio and Xcode. Licensing costs vary, and development teams need proficiency in Java/Kotlin for Android and Swift/Objective-C for iOS. |

**Recommendations**

1. **Operating Platform:**

Linux is recommended for hosting the server due to its stability, scalability, and cost-effectiveness. For client-side, a combination of platforms (macOS, Windows, iOS, Android) ensures broad accessibility.

1. **Operating Systems Architectures:**

The recommended Linux operating system will be based on a LAMP stack (Linux, Apache, MySQL, PHP/Python/Perl). This architecture supports modular and scalable web applications, enabling efficient management of game states and user interactions.

1. **Storage Management:**

A distributed database system like MySQL or MongoDB is recommended for managing game data efficiently across multiple platforms. These databases offer scalability and high availability, essential for handling large volumes of concurrent users.

1. **Memory Management:**

Linux offers robust memory management techniques, such as virtual memory and paging, which will be beneficial for the server. Client-side memory management will depend on the respective operating systems (macOS, Windows, iOS, Android), with each platform providing its own mechanisms for efficient memory allocation and management.

**A. Considerations for Effective Memory Management:**

* Efficient Image Loading: Implementing lazy loading techniques to load images as needed rather than all at once can reduce memory usage and improve performance.
* Caching: Utilizing caching mechanisms to store frequently accessed images and data can speed up rendering times and reduce memory consumption.
* Memory Optimization Tools: Employing tools to monitor and optimize memory usage during development can help identify and fix memory leaks and other issues.

**B. Storage Management Considerations:**

* Estimating Storage Needs: With 200 high-definition images at approximately 8MB each, the total storage required just for images would be around 1.6GB. Additional storage will be needed for user data, game states, and other assets.
* Scalability: Using scalable storage solutions like cloud storage (AWS S3, Google Cloud Storage) can ensure that storage capacity can grow as needed.
* Redundancy and Backup: Implementing redundancy and backup solutions will protect against data loss and ensure high availability.

**5. Distributed Systems and Networks:**

To enable communication between various platforms, a RESTful API will be used. This ensures seamless interaction between the client and server, with considerations for network connectivity and data synchronization. Using technologies like WebSockets can provide real-time updates and enhance the user experience.

**6. Security:**

User data protection will be ensured through encryption protocols like SSL/TLS. Additionally, regular security audits and compliance with data protection regulations (e.g., GDPR) will be implemented to safeguard user information across platforms. Authentication and authorization mechanisms, such as OAuth 2.0 and JWT, will be used to ensure secure access to the game.

**Comparison of Memory and Storage Management:**

**Memory Management:**

* Purpose: Manages the temporary data required for the application to run efficiently, such as rendering images and processing user interactions.
* Techniques: Includes virtual memory, paging, caching, and efficient image loading.
* Impact on Performance: Directly affects the speed and responsiveness of the application. Proper memory management ensures smooth gameplay and a better user experience.

**Storage Management:**

* Purpose: Manages the permanent data storage, including images, user data, and game states.
* Techniques: Includes estimating storage needs, using scalable storage solutions, and implementing redundancy and backup.
* Impact on Functionality: Ensures that all necessary data is available and can be accessed quickly. Proper storage management supports the scalability and reliability of the application.

*This evaluation provides a comprehensive guide for The Gaming Room to expand their application, Draw It or Lose It, into a web-based distributed environment. The recommendations aim to balance cost, scalability, and user accessibility, ensuring a successful platform expansion.*