

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | Alex Simonian | Initial draft |
| 2.0 | 12.01.2024 | Alex Simonian | Revision of Evaluation section |
| 3.0 | 12.13.2024 | Alex Simonian | Revision of Recommendations section |

**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 requested assistance in expanding their Android game "Draw It or Lose It" to a web-based platform. The game, which follows a similar format to the television game show "Win, Lose or Draw," needs to support multiple teams with multiple players while maintaining unique game and team identities.

Our solution implements a structured approach using Java, which aligns with their existing Android development. The design ensures single-instance game management and unique naming across the platform, providing a stable foundation for their web-based expansion.

## Requirements

Game Structure and Gameplay

* Support multiple concurrent teams in each game session
* Four one-minute rounds of gameplay per game
* Images must render progressively over 30 seconds
* Teams must have 60 seconds to guess the puzzle during their turn
* Non-playing teams get 15 seconds for one guess if active team fails
* Maintain a library of stock drawings for puzzle rendering

User Management

* Allow multiple players to be assigned to each team
* Support team creation and management
* Enable player assignment to teams
* Verify and enforce unique game and team names
* Track active players and teams during gameplay

Platform Requirements

* Develop a web-based version of the existing Android game
* Support cross-platform accessibility
* Ensure consistent gameplay experience across different platforms

## [Design Constraints](#_2et92p0)

The primary constraints for this web-based development include:

- Maintaining compatibility with the existing Android application

- Ensuring consistent gameplay experience across web browsers

- Managing unique game, team, and player instances

- Supporting real-time gameplay across different platforms

- Handling stock drawing storage and delivery efficiently

Java has been selected as the development language due to its compatibility with the existing Android codebase and robust web development capabilities.

## [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 application is structured using a main driver class that initiates games, teams, and players through the GameService class. The design implements several key features:

GameService Class:

- Uses singleton pattern to ensure single instance

- Controls game creation and management

- Maintains unique identifiers for games, teams, and players

Entity Relationships:

- Entity serves as base class for Game, Team, and Player

- Game contains multiple teams

- Teams contain multiple players

- All classes inherit basic attributes (id, name) from Entity

The design implements key object-oriented principles:

- Inheritance through the Entity class hierarchy

- Encapsulation via protected attributes and methods

- Singleton pattern in GameService

- Iterator pattern for unique name validation

**"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** | |  | | --- | | Mac servers are reliable for web applications, with strong design and good performance. However, they come with significant drawbacks. They are expensive compared to alternatives, and fewer hosting providers support Mac servers. Support and troubleshooting can take longer, which might increase operational costs and response times for The Gaming Room. |  |  | | --- | |  | | Linux emerges as the most compelling server-side solution. It is free, highly reliable, and widely supported for web hosting. Linux works seamlessly with numerous web tools and is extremely popular among developers for hosting web applications. While it might require more technical knowledge to set up initially, its flexibility, cost-effectiveness, and robust performance make it an excellent choice for scaling a web-based game platform. | |  | | --- | | Windows servers are user-friendly and integrate well with Microsoft tools. They are a solid choice for software built with Microsoft technologies. However, they are more expensive than Linux and may not handle heavy workloads as efficiently. They might require more resources and could potentially increase hosting costs for The Gaming Room. |  |  | | --- | |  | | |  | | --- | | Mobile devices are not suitable for hosting web applications. They have limited power, storage, and are better suited as clients connecting to servers rather than hosting server-side software. |  |  | | --- | |  | |
| **Client Side** | |  | | --- | | Building apps for Mac requires specialized skills and Apple's development software. The cost can be high since Apple devices are needed for testing and building. However, the development tools are high-quality and relatively easy to use once learned. The primary challenge is the specialized ecosystem and higher entry barriers. |  |  | | --- | |  | | |  | | --- | | Linux app development primarily focuses on web browser applications rather than standalone apps. Fewer people use Linux desktops, making it a less priority platform. Developing for Linux is cheaper but requires technical knowledge. Web-based approaches are typically more practical for Linux compatibility. |  |  | | --- | |  | | |  | | --- | | Windows is the most used desktop platform, making it crucial to support. There are good tools for building Windows apps, and the development costs are reasonable. Testing and ensuring compatibility with Windows features is essential, which might require additional development effort. |  |  | | --- | |  | | |  | | --- | | Mobile app development for iOS and Android requires significant investment. Building apps for multiple platforms takes more time and money. Cross-platform tools can help, but thorough testing on real devices is still necessary to ensure a consistent user experience. |  |  | | --- | |  | |
| **Development Tools** | |  | | --- | | Mac development uses Xcode, Apple's official software development tool. There are also cross-platform tools available. The ecosystem is robust but can be expensive and requires specialized knowledge. |  |  | | --- | |  | | |  | | --- | | Linux developers often use free tools like programming editors and web development tools. Linux is excellent for creating cross-platform applications but requires familiarity with command-line tools. |  |  | | --- | |  | | |  | | --- | | Windows offers strong development tools like Visual Studio, which makes app creation easier. There are also tools for building cross-platform applications. The development environment is user-friendly and well-supported. |  |  | | --- | |  | | Mobile development uses platform-specific tools like Android Studio and Xcode, as well as cross-platform frameworks like Flutter and React Native. These tools can help create apps that work on multiple platforms, potentially reducing development time and cost. |

## 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**: Based on the evaluation, Linux is the most suitable server platform for hosting "Draw It or Lose It." It is cost-effective, reliable, and widely supported by cloud providers. Its open-source nature ensures flexibility and scalability as the game grows. For client applications, using React.js for the web and Flutter or React Native for mobile platforms will enable cross-platform compatibility, reducing development time and costs.
2. **Operating Systems Architectures**: Linux’s monolithic kernel architecture supports high performance and efficient resource management, making it ideal for backend operations. Its ability to handle multiple tasks concurrently ensures the game can manage real-time interactions. On the client side, a web-based architecture using HTML, CSS, and JavaScript ensures platform independence, while mobile-specific architectures (iOS and Android) provide optimal app performance.
3. **Storage Management**: PostgreSQL is recommended as the primary database system due to its robust data integrity and scalability. It can store game, team, and player data while ensuring unique identifiers through constraints. Cloud storage services like AWS S3 or Google Cloud Storage can be integrated for storing game assets, supporting seamless scaling.
4. **Memory Management**: Linux employs advanced memory management techniques such as virtual memory, paging, and caching, ensuring efficient resource allocation during peak game activity. The recommended development frameworks (React.js and Flutter) also have built-in memory optimization features that help manage client-side memory effectively.
5. **Distributed Systems and Networks**: To enable cross-platform communication, the system should use REST APIs for standard data exchanges and WebSockets for real-time communication. Implementing a load balancer will distribute traffic across multiple servers, ensuring high availability and reliability. Cloud-based services with Content Delivery Networks (CDNs) will further optimize performance by delivering content from geographically closer servers.
6. **Security**: Security is crucial to protect user data. Implementing HTTPS and TLS protocols will secure data transmission. OAuth2 or similar authentication services will manage secure user logins. PostgreSQL should have role-based access control and encrypted data storage. Regular security audits, penetration testing, and compliance with data privacy standards like GDPR will further enhance protection across all platforms.