

Draw It or Lose It

# CS 230 Project Software Design Template

Version 2

# Table of Contents

[CS 230 Project Software Design Template 1](#_bookmark0)

[Table of Contents 2](#_bookmark1)

[Document Revision History 2](#_bookmark2) [Executive Summary 3](#_bookmark3)

[Requirements 3](#_bookmark4)

[Design Constraints 3](#_bookmark5)

[System Architecture View 4](#_bookmark6) [Domain Model 4](#_bookmark7) [Evaluation 5](#_bookmark8)

[Recommendations 8](#_bookmark9)

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | <01/26/2025> | Eddy Kwon | Initial version created for CS 230 Project one |
| 1.1 | <02/09/2025> | Eddy Kwon | Updated for CS 230 Project Two with platform  evaluations and recommendations |
| 2.0 | <02/23/2025> | Eddy Kwon | Completed CS 230 Project Three with platform analysis, storage, memory, distributed systems, and security recommendations. |

**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

# The Gaming Room, which currently has an Android version of the game "Draw It or Lose It," seeks to expand the game to multiple platforms. The goal is to increase the game's accessibility and usability, by offering it on web browsers and other mobile devices like iOS. The core challenge is ensuring scalability, multi-user support, and maintaining high performance across all platforms, all while adhering to robust security standards. The solution is to design the game using Object-Oriented Programming principles, ensuring modularity and scalability across various operating systems, including Linux, Windows, Mac, and mobile devices.

# Requirements

# *Requirements*

# *Business Requirements:*

# Multi-Team Support: The game should allow multiple teams to play against each other.

# Cross-Platform: The game must be web-based and accessible across multiple platforms (Linux, Mac, Windows, and mobile).

# Unique Identifiers: Each game instance, team, and player should have a unique identifier to ensure smooth gameplay.

# Scalability: As the number of teams and players grows, the system should be able to handle the increased load.

# *Technical Requirements:*

# Object-Oriented Design: Use Object-Oriented Programming (OOP) principles to model key entities such as Game, Team, and Player.

# Singleton Pattern: Implement a Singleton pattern for the GameService class to manage all instances of the game.

# Scalability: The system should allow multiple game instances without performance degradation.

# Performance Across Platforms: Ensure optimal performance on mobile and web platforms under various network conditions.

# Player Authentication: Implement a system where players can register, log in, and authenticate to ensure data consistency and security across sessions.

# Design Constraints

1. Web-Based Application: The game must be designed as a responsive, web-based application, ensuring compatibility across browsers like Chrome, Firefox, and Safari.
2. Single Instance for Each Game: The game must ensure only one instance runs at any given time, using the Singleton pattern.
3. Scalability: The system must handle a growing number of players and game instances without performance issues.
4. Data Persistence and Synchronization: A cloud-based solution like Firebase should be used for real-time data synchronization and persistent storage across platforms.

# System Architecture View

# The system will include a layered architecture with a frontend client-side (accessible through web browsers and mobile devices) and a backend server-side for handling game logic, data storage, and communication between players. The backend should use web technologies like Node.js and databases such as Firebase for storage and authentication.

# Domain Model

The "Draw It or Lose It" game involves several key classes, including:

* **Game:** Manages game instances, ensuring one instance exists at a time.
* **Team:** Represents the teams in the game, including multiple players.
* **Player:** Represents individual players within a team.
* **GameService:** Manages the overall game state and ensures game integrity.

"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

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** | Macs are good for smaller-scale applications but not ideal for enterprise-level hosting. | Linux is highly cost-effective, secure, and the preferred choice for web servers. | Windows is user-friendly, ideal for enterprise systems but resource-heavy. | Mobile devices are not suitable for hosting; they are only client-side. |
| **Client Side** | Macs offer performance optimization for iOS and macOS apps but may incur higher costs. | Linux offers flexibility and cost-effectiveness for development. | Windows supports many client-side applications and is highly compatible with various software. | Mobile devices (iOS/Android) are ideal for client-side interactions but not for hosting. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Tools** | Xcode is the primary tool, but it's more suited for Apple environments. | Linux supports open-source tools like Eclipse and Visual Studio Code. | Windows supports Visual Studio, Eclipse, and many other IDEs. | Mobile platforms need Android Studio and Xcode for development. Cross-platform tools like React Native or Flutter help manage complexity. |

# Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

# Operating Platform: For expanding "Draw It or Lose It" to other platforms, I recommend using Linux-based servers for the backend. Linux offers stability, cost-effectiveness, and excellent support for web-based applications, making it ideal for scaling. For client-side development, React Native or Flutter is recommended for seamless cross-platform compatibility across iOS and Android devices.

# Operating System Architectures: Windows Server edition should be used for hosting the backend, as it integrates well with existing Microsoft tools and supports various server-side applications. For mobile platforms, both iOS and Android are ideal for running the game on mobile devices due to their user-friendly ecosystems and strong app store support.

# Storage Management: A simple cloud-based storage system like Google Drive or Dropbox can store small amounts of data. However, as the game scales, more robust systems such as Amazon S3 or Google Cloud Storage should be considered for managing large volumes of player data and backups.

# Memory Management: Windows provides robust memory management tools, such as automatic memory allocation and garbage collection. For mobile platforms, memory is managed by the operating system to ensure efficient use, removing unnecessary data when the app is not in use.

# Distributed Systems and Networks: To allow seamless communication across devices, HTTP requests should be used to send data between the server and the app. This allows the game to handle player interactions and data synchronization efficiently. To minimize the impact of network outages, the system should have reconnection capabilities to ensure that players can continue without losing data.

# Security: Security is a high priority for the client. HTTPS should be implemented for secure communication, and mobile apps should utilize Keychain (iOS) or Keystore (Android) to protect user credentials. Additionally, Role-Based Access Control (RBAC) should be used to ensure that users only have access to data appropriate to their roles.