

DRAW IT OR LOSE IT

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 2.0 | 12/21/2024 | Corderro Artz | Second iteration of the software design document. |

**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’s game application, **Draw It or Lose It**, is currently available only on Android devices. The company aims to expand the game to multiple platforms, including desktop operating systems (Linux, Mac, and Windows) and mobile devices (Android and iOS). This document evaluates the characteristics, advantages, and weaknesses of each platform for both server- and client-side deployment. It also provides recommendations for the most suitable operating platform, system architecture, and tools required for development.

The solution proposed involves leveraging a web-based distributed environment to ensure scalability, platform independence, and cost-effectiveness. By utilizing a responsive HTML5 interface for the client-side and a robust server-side implementation, the game can support thousands of players across various devices. This document outlines the technical requirements, constraints, and recommendations to guide The Gaming Room in achieving its goals.

## Requirements

The primary business and technical requirements for the project include:

1. ***Cross-Platform Compatibility****: The game must be accessible on mobile devices (Android and iOS) and desktop operating systems (Linux, Mac, Windows) using a responsive web-based interface.*
2. ***Scalability****: The solution must support thousands of simultaneous players with minimal latency.*
3. ***Cost Efficiency****: Licensing and development costs must be minimized without sacrificing performance or security.*
4. ***Security****: User data and gameplay must be protected through secure authentication and data transmission protocols.*
5. ***User Experience****: The game must provide a seamless and responsive experience across all platforms.*

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment introduces the following design constraints:

1. **Platform Independence**: The client-side application must function on various platforms without requiring native apps for each environment. This necessitates responsive web design and extensive browser compatibility testing.
2. **Server Scalability**: The server-side must handle an increasing number of users and requests efficiently. This requires selecting a suitable operating platform capable of supporting high availability and load balancing.
3. **Development Time and Expertise**: Supporting multiple platforms increases complexity, requiring skilled developers proficient in web technologies and cross-platform tools.
4. **Security Requirements**: Protecting user data across distributed systems requires robust encryption, secure authentication mechanisms, and protection against network vulnerabilities.

These constraints influence the choice of tools, frameworks, and platforms used to develop and host the application.

## [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 provided UML diagram represents the structure of the game application, illustrating the relationships between core classes.

1. **Entity**: Acts as a base class for Game, Team, and Player, providing shared attributes like id and name. This demonstrates inheritance, a key object-oriented programming principle.
2. **GameService**: Implements the Singleton design pattern to ensure only one instance of the service exists. This class manages the lifecycle of games, teams, and players, enforcing data consistency.
3. **Relationships**:
   * The GameService manages multiple Game objects (0..\* relationship).
   * Each Game contains multiple Team objects (0..\* relationship).
   * Each Team is composed of multiple Player objects (0..\* relationship).

This design efficiently organizes the application’s data and ensures scalability and maintainability.

**"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** | macOS supports server hosting but lacks flexibility compared to Linux. Licensing and hardware costs are high. Best suited for small-scale deployments. | Linux is the most cost-effective and scalable option for server hosting. It is open-source, highly reliable, and widely supported by cloud platforms. | Windows Server provides robust enterprise features but comes with high licensing costs. It requires additional security management | Mobile devices are unsuitable for production server hosting. They are useful for testing purposes only, given their limited power. |
| **Client Side** | Development for macOS requires responsive web apps compatible with Safari. Xcode is essential. Cost and time are higher due to Apple hardware requirements. | Linux supports modern web browsers like Firefox and Chrome, making it ideal for testing and client-side deployment. Development is cost-effective. | Windows supports a wide range of browsers (Edge, Chrome). Development is straightforward, but testing for compatibility across versions is necessary. | Mobile development requires expertise in both Android (Kotlin/Java) and iOS (Swift). Testing touch interfaces and screen responsiveness is critical. |
| **Development Tools** |  |  |  |  |
| **Languages & Tools** | Swift, JavaScript, HTML/CSS. IDE: Xcode, VS Code. Testing: Safari Developer Tools. | JavaScript, Python, PHP, Java. IDEs: Eclipse, IntelliJ IDEA, VS Code. Testing: Selenium. | C#, JavaScript, HTML/CSS. IDE: Visual Studio, VS Code. Testing: Edge Developer Tools. | Android: Java/Kotlin, Android Studio. iOS: Swift, Xcode. Cross-platform frameworks like Flutter or React Native. |
| **Impact on Development** | Requires Apple hardware, increasing costs. Xcode simplifies iOS-specific development but limits flexibility. | Open-source tools reduce development costs. Linux is ideal for web development and testing due to its stability. | Development is straightforward with Visual Studio but requires licensing. Windows is widely used in enterprises, ensuring developer availability. | Developing for both Android and iOS increases time and expertise requirements. Cross-platform frameworks can reduce duplication of effort. |
| **Licensing Costs** | High cost for macOS hardware. Xcode is free | Most tools are free and open-source. Optional paid enterprise support. | Visual Studio may require a subscription. Windows Server has high licensing costs. | Android Studio is free. Xcode is free, but Apple hardware is required. Cross-platform tools may have subscription fees. |

## Recommendations

Based on the evaluation, the following recommendations are made:

1. **Operating Platform**:
   * Use **Linux** for server-side hosting due to its cost-effectiveness, reliability, scalability, and compatibility with cloud platforms.
   * For client-side development, ensure compatibility with all major web browsers across Linux, Mac, Windows, and mobile devices.
2. **Operating System Architectures**:
   * The recommended architecture is a web-based distributed environment, utilizing Linux for the server and responsive HTML5 for the client.
3. **Storage Management**:
   * Use cloud-based storage solutions like AWS S3 or Google Cloud Storage for scalability and reliability.
4. **Memory Management**:
   * Linux efficiently handles memory management for server applications, using techniques like paging and caching to optimize performance.
5. **Distributed Systems and Networks**:
   * Communication between platforms will be achieved using a RESTful API. Network dependencies will be minimized through load balancing and redundancy.
6. **Security**:
   * Implement HTTPS for secure communication. Use OAuth 2.0 or JWT for user authentication. Protect data at rest with encryption and ensure regular security updates.