

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | <05/11/1999> | <Blake Snyder> | <Brief description of changes in this revision> |

**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 seeks to make a web-based version of their existing game, draw it or Lose it, to be accessible across multiple Platforms. The game involves teams competing to guess drawn images in a set time. The web-based version must be able to host multiple teams, ensure unique games and team names, and allow one instance of the game at a time. To meet these requirements a web-based architecture will be implemented. The application will be developed using scalable framework, this ensures responsiveness and cross-platform compatibility. The use of unique identifiers will prevent duplicate game or team names.

## Requirements

Business requirements are developing a web-based version of Draw it or lose it, Ensure real time gameplay and responsiveness across multiple devices, and maintain an intuitive user interface for players to join teams and participate in.  
  
Technical requirements include supporting multiple teams within a game session, allow multiple players to join and team, Enforce unique game and team names, and Utilize OOP principles for modularity and scalability.

## [Design Constraints](#_2et92p0)

Design constraints would include Scalability, Cross-platform Compatibility, and data integrity. The server should be able to handle varying player loads without performance issues. The game must function on multiple platforms requiring a responsive design. Unique identifier for games, teams, and players will ensure no duplication issues will arise.

## [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 UML class diagram for the application follows an object-oriented structure featuring key classes:  
1. Entity (base class): provides a unique id and name   
2. GameService (singleton pattern): Manages all active games, players and teams. Ensures only one instance of game exists.  
3. Game Class: Maintains a list of teams and enforces unique team names

4. Team Class: Handles team specific logic and ensures unique team names

5. Player Class: Represents individual user

**"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** | Stable but limited cloud options; not as widely used for server hosting. | Reliable, widely used for web servers. Best choice for hosting. | Supported but requires additional setup for performance optimization. | Not suited for server hosting. |
| **Client Side** | Supports modern web browsers and game engines. | Fully supports web browsers but may have limited graphical acceleration. | Strong compatibility with various game frameworks. | Requires responsive UI and touch-based controls. |
| **Development Tools** | Xcode, Visual Studio Code, WebStorm | VS Code, Eclipse, IntelliJ | Visual Studio, JetBrains IDEs, WebStorm | Android Studio, Xcode (iOS), React Native, Unity |

## 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**:

My recommendation is a Linux based cloud platform. I recommend this platform because of its Reliability and scalability, Cost effectiveness, security, and cross-platform compatibility. Linux is widely used in cloud environments and supports scalable hosting solutions. Using Open-Source Linux distributions reduce licensing costs. Linux has strong security mechanisms, reducing vulnerabilities in online applications. The web-based solution ensures compatibility with Windows, macOS, and mobile devices.

1. **Operating Systems Architectures**:   
   For operation system architecture I recommend a Linux-based distributed architecture with microservices. This is the correct choice because its client server model, load balancing, and microservices. A web-based frontend communicates with a backend server hosted on the cloud. It distributes traffic efficiently to handle multiple concurrent players. Separate modules for game logic, user authentication, and real-time communication allow for modular and efficient scaling.

**Storage Management**:   
For storage I recommend a cloud base NoSQL database for high performance, Scalability, schema flexibility, and reliability. NoSQL databases are optimized for real-time applications with fast reads/writes. They support large scale user interactions without performance degradation. It allows dynamic storage of game session data, team structures, and player scores. Cloud-based storage solutions provide backup and failover mechanisms.

1. **Memory Management**:   
   For Memory Management I recommend a managed memory allocation with caching using Automatic Memory Management, Caching for performance. The Linux OS uses virtual memory and garbage collection to optimize memory usage. a caching layer stores frequently accessed game data, reducing database queries.
2. **Distributed Systems and Networks**:   
   For Distributed systems and networks I recommend cloud base backend. Cloud based backend has Load balances for distributing network traffic to prevent server overload. It also has Backup systems to ensure game sessions are preserved even if the server crashes.
3. **Security**:  
   Recommendations for security are end-to-end encryption, User authentication, and cloud security. End to end encryption protects user information being sent. User authentication ensures secure logins, and cloud security helps against DDOS attacks.