

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 | 01/23/2025 | Nicole Hutto | Initial draft for The Gaming Room Project |
| 2.0 | 02/06/2025 | Nicole Hutto | Update for The Gaming Room Project |
| 3.0 | 02/18/2025 | Nicole Hutto | Updating Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room (Client) wants to expand its game, “Draw It or Lose It,” to support web-based platforms. The current game is strictly only available on the Android Play Store. The proposed solution will make it available across multiple platforms and Play Stores.

This document outlines the software design, addressing the Client's requirements, constraints, and objectives.

## Requirements

1. Web-based Platform: The game must be accessible through a web-based platform compatible with various devices and operating systems.
2. Multi-Team Support: Each game should allow one or more teams to participate, with the ability to assign players to each team working on the project.
3. Singleton Instance: The application must ensure that only one instance of the game exists in memory at any time. This is achieved by using multiple and unique identifiers for games, teams, and players.
4. Game Rounds: The game consists of multiple rounds (up to four), each lasting one minute. Drawings are revealed progressively, becoming complete at the 30-second mark.
5. Guessing Mechanism: If a team fails to guess the puzzle within the one-minute time limit, the other teams will get a fifteen-second window to guess the puzzle before moving on to the next team, which will also get a fifteen-second window.

## [Design Constraints](#_2et92p0)

The proposed application must adhere to the following constraints:

* Support one or more teams in the game, which would have multiple players simultaneously.
* Ensure the game and team names are unique.
* Limit to a single game instance in memory at any given time.
* Adapt the existing Android-based codebase for a web-based cross-platform environment.
* Implement in Java using design patterns to enforce these requirements.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: Nothing is required here for these projects, but this section serves as a reminder that describing the application's system and subsystem architecture, 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 domain model for the game consists of several classes.

* Entity Class: This serves as the base class on which everything runs and provides shared attributes like ‘id’ and ‘name.’
* Game Class: Manages the teams, allowing the addition of unique names.
* Team Class: Manages players and ensures unique player names within each team.
* Player Class: Represents individual participants in the game.
* GameService Class: Implements the Singleton design pattern, ensuring only one instance exists to manage the game lifecycle.
* ProgramDriver Class: Acts like the entry point, interacting with the GameService

**"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** | High security, user-friendly, but costly & limited scalability. | Open-source, highly secure, requires expertise & has a steep learning curve. | Best for gaming is easy to use and has excellent software support, but it has licensing costs. | Broad accessibility but requires cross-platform compatibility. |
| **Client Side** | Seamless integration with the Apple ecosystem but requires paid licenses. | Cost-effective but complex; debugging can be challenging. | Most effortless for developers, strong support & tools. | Requires frameworks like React Native or Flutter for cross-platform support. |
| **Development Tools** | Swift, Objective-C, JavaScript. IDEs: Xcode, VS Code. | Python, Java, JavaScript. IDEs: Eclipse, VS Code, PyCharm. | C#, Java, JavaScript. IDEs: Visual Studio, IntelliJ. | Kotlin (Android), Swift (iOS), JavaScript. IDEs: Android Studio, Xcode. |

## Recommendations

**Operating Platform**:

* Use a Linux-based cloud server for scalability, security, and cost-effectiveness
* Utilize AWS or Google Cloud for high availability and performance

**Operating Systems Architecture**:

• Implement a microservices-based architecture for flexibility

* Use Docker containers and Kubernetes for efficient multi-platform deployment

**Storage Management**:

* Use cloud-based storage (AWS S3, Google Cloud) for scalability and reliability
* Ensure automatic backups and global accessibility for secure data storage

**Memory Management**:

* Leverage virtual memory techniques like paging and segmentation for system stability
* Use catching solutions (e.g., Redis) to improve response times and reduce latency

**Distributed Systems & Networks**:

* Use a centralized server for real-time synchronization

• Implement WebSockets for instant player interactions

* Utilize load balancing and failover strategies to handle connectivity issues

**Security**:

• Encrypt data transmissions using TLS/SSL

* Implement multi-factor authentication(MFA) for account security
* Use role-based access control (RBAC) to limit user permissions
* Regularly update security patches to prevent vulnerabilities

By following these recommendations, The Game Room can successfully expand “Draw IT or Lose It” across multiple platforms, ensuring a smooth, secure, and scalable gaming experience.