Joshua Hale

July 11, 2024

CS-230

3-1 Project One



# < 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 4**

[**Evaluation 5-6**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | July 11, 2024 | Joshua Hale | Initial version for The Gaming Room |

**Executive Summary**

The Gaming Room aims to expand its game, Draw It or Lose It, from an Android app to a web-based platform supporting multiple devices. Draw It or Lose It is a game where teams compete to guess images rendered from a stock library. The application needs to handle multiple teams and players, ensuring unique names and maintaining a single instance of the game in memory. This document outlines the software design to meet these requirements using best practices in software engineering, including the singleton and iterator design patterns.

## 

**Requirements**

The game application must fulfill the following requirements:

* A game can have one or more teams.
* Each team can have multiple players.
* Game and team names must be unique.
* Only one instance of the game should exist in memory at any given time.

**Design Constraints**

Developing the game in a web-based distributed environment introduces several constraints:

1. **Scalability**: The system must handle multiple concurrent users and teams.
2. **Consistency**: Unique identifiers for games, teams, and players must be maintained across sessions.
3. **Performance**: The game must render images and process guesses in real-time.
4. **Security**: User data and game integrity must be protected.

These constraints necessitate careful design choices, such as using a singleton pattern to ensure only one instance of the game service exists and using efficient data structures to manage game state and user interactions.

## [System Architecture View](#_ilbxbyevv6b6)

*(Not required for this project)*

**Domain Model**

The UML diagram represents the key components and their relationships in the game application. The ‘Entity’ class serves as a base class for ‘Game’, ‘Team’, and ‘Player’, encapsulating common attributes like ‘id’ and ‘name’. The ‘GameService’ class uses the singleton pattern to ensure a single instance manages all game-related operations. The ‘Game’ class can have multiple Team instances, and each Team can have multiple Player instances. Object-oriented principles like inheritance and encapsulation are utilized to manage common behaviors and attributes, enhancing code reuse 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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Offers a stable development environment with excellent GUI tools, but limited server deployment compared to Linux. | Highly suitable for web servers due to its stability, security, and performance; vast community support. | Provides a robust environment with extensive enterprise support but may require more resources and licensing costs. | Limited as a server but crucial for client-side interaction; need to ensure compatibility and performance. |
| **Client Side** | High-quality user experience and development tools but higher cost. | Flexible and customizable but can be complex for non-technical users. | Widely used with comprehensive support for various applications; cost considerations. | Essential for accessibility; requires responsive design and efficient resource management. |
| **Development Tools** | * Xcode * IntelliJ IDEA * Eclipse | * Eclipse * IntelliJ IDEA * NetBeans | * Visual Studio * IntelliJ IDEA * Eclipse | * Android Studio * Xcode for iOS * Cross-platform tools like Flutter. |

**Recommendations**

1. **Operating Platform:**

Linux is recommended for its stability, performance, and cost-effectiveness for server deployment.

1. **Operating Systems Architectures:** The Linux platform's architecture supports modular and scalable web applications.
2. **Storage Management:** Utilize a robust database management system like MySQL or PostgreSQL to handle game data.
3. **Memory Management:** Implement efficient caching strategies and manage memory allocation to ensure real-time performance.
4. **Distributed Systems and Networks:** Use RESTful APIs and WebSocket protocols to enable communication between platforms, ensuring data consistency and real-time updates.
5. **Security:** Employ encryption, secure communication channels (HTTPS), and regular security audits to protect user information and game integrity.