
Draw It or Lose It Web Application

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**



## Document Revision History

| Version | Date | Author | Comments |
|---|---|---|---|
| 1.0 | 05/21/2024 | Tashyra Adams | Initial version 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

The Gaming Room aims to expand their existing Android game, Draw It or Lose It, into a web-based application that supports multiple platforms. The game involves teams guessing a puzzle based on images drawn within a minute. The proposed solution involves developing a web-based application with a robust backend to manage games, teams, and players, ensuring unique names and identifiers. Key design patterns such as Singleton and Iterator will be employed to streamline the development and maintain efficient and reliable software. This document outlines the design constraints, system architecture, domain model, and provides evaluations and recommendations for the project.

## Requirements

- **Business Requirements:** The game should support multiple teams and players, ensure unique game, and team names, and allow only one instance of the game in memory.
- **Technical Requirements:** Implement the Singleton pattern for game instance management and the Iterator pattern for managing game and team names. The application should be web-based and accessible across multiple platforms.

## Design Constraints

1. **Concurrency:** The application must handle multiple users accessing and updating game data simultaneously, requiring efficient synchronization mechanisms.
2. **State Management:** Maintaining the state of the game across sessions and ensuring data consistency is challenging in a distributed environment.
3. **Scalability:** The application must scale to support an increasing number of users and teams without performance degradation.
4. **Security:** Protecting user data and ensuring secure communication between clients and the server is critical.

## System Architecture View

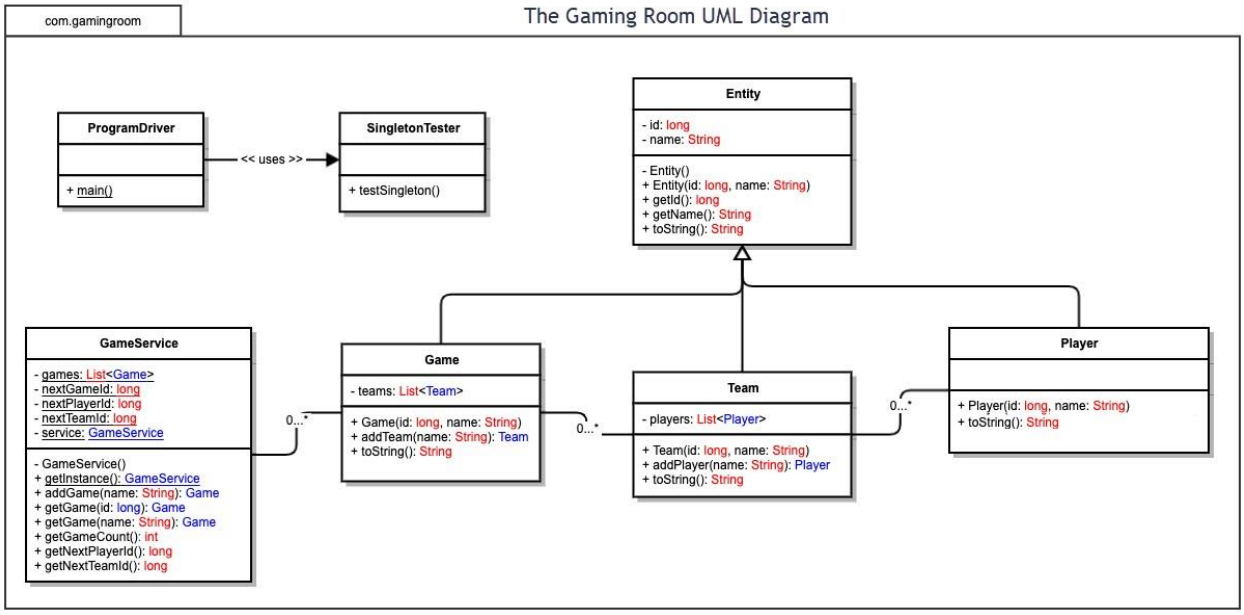
Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## Domain Model

- **Entity Class:** Serves as a base class with common attributes (**id** and **name**) and methods.
- **GameService Class:** Manages game instances, ensuring a single instance using the Singleton pattern.
- **Game Class:** Inherits from **Entity**, manages teams within the game.
- **Team Class:** Inherits from **Entity**, manages players within the team.
- **Player Class:** Inherits from **Entity**, represents individual players.
- **ProgramDriver Class:** Contains the main method to test the application.
- **SingletonTester Class:** Tests the Singleton implementation.

Object-oriented principles demonstrated:

- **Inheritance: Game**, **Team**, and **Player** classes inherit from **Entity**, promoting code reuse.
- **Encapsulation:** Each class manages its data and provides methods to access and modify it.
- **Singleton Pattern:** Ensures only one instance of **GameService** exists, managing game state centrally.




## 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** | Mac servers are stable and reliable but more expensive. | Linux servers are highly customizable, cost-effective, and widely used. | Windows servers offer good support and ease of use but can be costly. | Mobile devices are not typically used for hosting servers. |
| **Client Side** | Development on Mac requires higher initial cost but is well-supported for graphics. | Linux development is cost-effective with many open-source tools but may require more expertise. | Windows development is user-friendly with extensive tool support. | Mobile development requires optimizing for various screen sizes and touch interfaces. |
| **Development Tools** | Xcode and Swift are primary tools for Mac development. | Linux uses GCC, Clang, and various IDEs like Eclipse and NetBeans. | Visual Studio and .NET framework are commonly used for Windows. | Android Studio and Xcode are used for mobile development. |

**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**: Recommend using Linux for the server-side due to its cost-effectiveness, stability, and flexibility. For client-side, support should be provided for Windows, Mac, and mobile devices to ensure broad accessibility.

2. **Operating Systems Architectures**: Linux architecture should be leveraged for the server due to its efficiency in handling web applications. On the client side, the application should be designed to run on Windows, MacOS, and mobile operating systems (iOS and Android).

3. **Storage Management**: Use a relational database such as MySQL or PostgreSQL for storing game, team, and player data, ensuring ACID compliance and efficient querying.

4. **Memory Management**: Utilize efficient memory management techniques in the chosen operating platforms, leveraging garbage collection in Java for server-side operations.

5. **Distributed Systems and Networks**: Employ RESTful APIs for communication between the client and server. Use load balancers and distributed database systems to handle scalability and ensure reliable connectivity.

6. **Security**: Implement HTTPS for secure communication, use encryption for sensitive data, and ensure regular security audits. Employ authentication and authorization mechanisms to protect user data.