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	06/23/2024	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

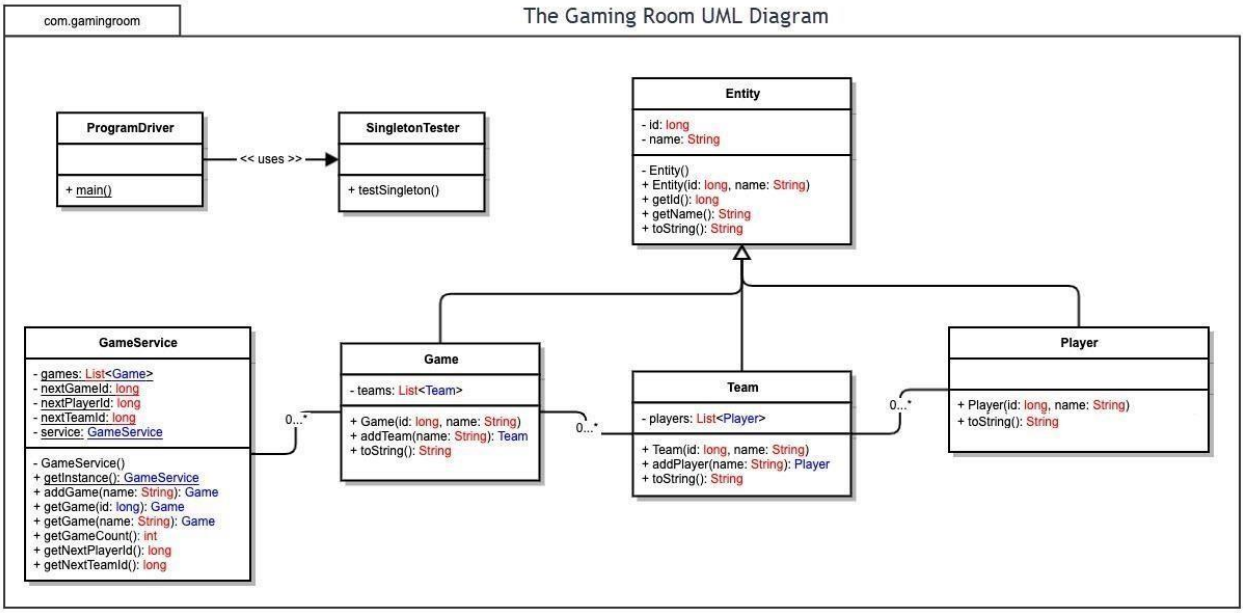
Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and 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. They are well-suited for environments requiring highperformance graphics processing.	Linux servers are highly customizable, costeffective, and widely used. They are preferred for their stability, security, and scalability.	Windows servers offer good support and ease of use but can be costly. They are commonly chosen for environments with extensive Microsoft infrastructure.	Mobile devices are not typically used for hosting servers.
Client Side	Development on Mac requires higher initial cost but is well-supported for graphics. Mac users often have higher purchasing power, making them an attractive demographic for premium applications.	Linux development is cost-effective with many open-source tools but may require more expertise. Linux users value freedom and privacy, making them receptive to applications aligned with open-source principles.	Windows development is user-friendly with extensive tool support. Windows users represent a large market share, making it essential for applications targeting mainstream audiences.	Mobile development requires optimizing for various screen sizes and touch interfaces. Mobile users expect seamless performance and intuitive user interfaces tailored to their devices.
Development Tools	Xcode and Swift are primary tools for Mac development. They offer a streamlined development experience with deep integration into macOS and iOS ecosystems.	Linux uses GCC, Clang, and various IDEs like Eclipse and NetBeans. They provide powerful development capabilities for building software across diverse Linux distributions.	Visual Studio and .NET framework are commonly used for Windows. They offer comprehensive development tools and extensive libraries for building Windows applications.	Android Studio and Xcode are used for mobile development. They provide robust development environments with tools for designing, debugging, and testing mobile applications.

Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

Operating Platform

For hosting the Draw It or Lose It game across various computing environments, I recommend leveraging **Linux** as the operating platform. Linux offers several advantages:

- **Cost-effectiveness:** It is open-source and does not incur licensing costs.
- **Customizability:** Tailor the environment to specific game requirements.
- **Stability:** Known for reliability in server environments, crucial for hosting game instances simultaneously.

Operating Systems Architectures

Linux utilizes a **monolithic architecture**. This architecture is advantageous for Draw It or Lose It due to:

- **Efficiency:** Direct access to kernel functions improves performance.
- **Scalability:** Supports multiple processes and threads efficiently.
- **Security:** Reduced layers simplify security management, enhancing robustness against threats.

Storage Management

Implement **MySQL** as the database management system (DBMS) for storing game data:

- **Reliability:** ACID compliance ensures data integrity.
- **Scalability:** Supports large datasets and concurrent users.
- **Performance:** Indexing and query optimization for quick data retrieval.

Memory Management

Linux employs **virtual memory** management techniques:

- **Demand Paging:** Efficiently manages memory resources by loading pages as needed.
- **Caching:** Utilizes cache memory to reduce disk I/O, optimizing game performance.
- **Memory Protection:** Segmentation and paging mechanisms ensure data integrity and security.

Distributed Systems and Networks

To enable communication between platforms, utilize **RESTful APIs** and **WebSocket protocols**:

- **Interoperability:** Facilitates real-time data exchange across different operating systems.
- **Fault Tolerance:** Redundancy in network connections ensures continuous service availability.
- **Load Balancing:** Distributes traffic across servers to prevent overload and enhance responsiveness.

Security

Implement robust security measures:

- **Encryption:** Use TLS/SSL protocols for secure data transmission.
- **Access Control:** Role-based access controls (RBAC) limit permissions based on user roles.
- **Auditing:** Regular security audits and logging to monitor and mitigate potential threats.
- **Firewall:** Configure firewall rules to protect against unauthorized access.