The Game Room
CS 230 Project Software Design
Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/6/2023	Robert Heavner	Initial Draft completed
1.1	11/20/2023	Robert Heavner	Revised the Evaluation section
1.2	12/04/2023	Robert Heavner	Recommendations

Executive Summary

The development of a web-based gaming application that makes managing games, teams, and players easier is the current software design challenge. The methodical approach that follows object-oriented design principles is the suggested remedy for building an efficient, scalable, and maintainable system. The design will accommodate different client needs, guaranteeing end users a smooth experience. The software will be built in a distributed environment that can dependably support simultaneous user interactions and data transfers.

Requirements

One of the client's business needs is a web-based application that can manage several teams, players, and games at once. The technical specifications place a strong emphasis on the need for a system that can handle big datasets quickly and with little latency. For simplicity of use and navigation, the program must also guarantee data integrity and offer a simple user interface.

Design Constraints

There are certain design restrictions when creating a game application on a distributed web environment. These include: Data Integrity, Scalability, Concurrency and Response times.

System Architecture View

Please note: There is nothing required here for these projects.

Domain Model

ProgramDriver: The main method, which acts as the application's entry point, is contained in this class.

SingletonTester: Used to test the Singleton design pattern, likely ensuring that only one instance of a certain class is created within the application.

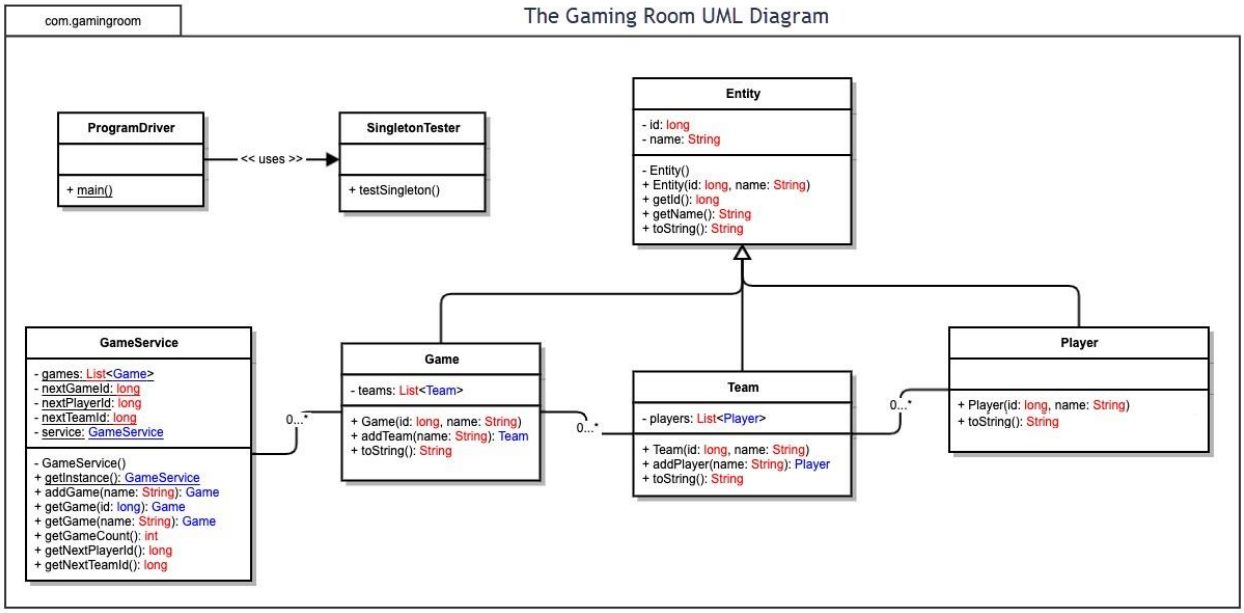
Entity: Acts as a foundation class from which additional specialized classes descend, with shared characteristics like name and id.

GameService: Manages game-related processes by putting the Singleton pattern into practice. It has functions for handling game, player, and team unique identifiers as well as adding and retrieving games.

Game: Represents a game and contains teams. It can add a new team to the game.

Team: Represents a team and contains players. It can add a new player to the team.

Player: Represents a player in the game.


com.gamingroom
The Gaming Room UML Diagram
ProgramDriver
+ main()
<< uses >>
SingletonTester
+ testSingleton()
Entity
- id: long
- name: String
- Entity()
+ Entity(id: long, name: String)
+ getId(): long
+ getName(): String
+ toString(): String
GameService
- games: List<Game>
- nextGameId: long
- nextPlayerId: long
- nextTeamId: long
- service: GameService
- GameService()
+ getInstance(): GameService
+ addGame(name: String): Game
+ getGame(id: long): Game
+ getGame(name: String): Game
+ getGameCount(): int
+ getNextPlayerId(): long
+ getNextTeamId(): long
0...*
Game
- teams: List<Team>
+ Game(id: long, name: String)
+ addTeam(name: String): Team
+ toString(): String
0...*
Team
- players: List<Player>
+ Team(id: long, name: String)
+ addPlayer(name: String): Player
+ toString(): String
0...*
Player
+ Player(id: long, name: String)
+ toString(): String

Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	MacOS servers offer a stable UNIX-based environment which is beneficial for hosting web applications	Linux is widely recognized for its strength, stability, and performance as a server operating system	Windows Server is known for its integration with other Microsoft products and services, which can be a significant advantage.	Mobile devices are not traditionally used for hosting web-based software applications.
Client Side	Developing client-side software for Macs often involves the use of Swift or Objective-C and can benefit from a user base that is typically willing to pay for quality software.	Supporting client-side software on Linux can be less costly due to the use of open-source tools and languages. However, it can be complex.	Developing for the Windows client side may involve a broader user base, which is an advantage.	Mobile client-side development requires expertise in mobile-specific programming languages such as Swift for iOS and Kotlin or Java for Android.
Development Tools	The primary tools for developing on Mac include Xcode for applications written in Swift or Objective-C. Licensing Costs: Xcode is free	Linux supports a wide array of development tools, including open-source IDEs and a variety of languages such as Python, PHP, Ruby, and JavaScript for web development. Licensing Costs: Mostly free due to being open source.	Development on Windows often involves Visual Studio for C# and .NET applications. Licensing Costs: Visual Studio has both free and paid versions; costs depend on the edition and organizational needs.	Mobile development utilizes different IDEs for different platforms: Xcode for iOS applications and Android Studio for Android applications. Licensing Costs: Android Studio is free; Xcode is free but needs Mac hardware. Additional costs for app store submissions and developer accounts.

Recommendations

1. **Operating Platform**: For The Gaming Room to expand "Draw It or Lose It" to various computer environments, a cross-platform operating system that supports a variety of devices, including desktops and mobile ones, is advised. A web-based solution utilizing JavaScript and HTML5 frameworks (like React or Angular) would be the best option given these specifications.

2. **Operating Systems Architectures**:
   The suggested architecture for the cross-platform environment includes:
   Client-Side: A responsive front-end that adjusts to desktop and mobile browsers.
   Server-Side: A Node.js server application that is compatible with Linux, macOS, and Windows server environments.
   Communication: A RESTful API or WebSocket communication protocol to facilitate real-time interaction between the client and server.
   Scalability: Containerization using Docker or orchestration with Kubernetes for easy scaling and deployment across various operating systems and cloud providers.
3. **Storage Management**: The suggested platform may benefit from a hybrid storage management system that combines NoSQL and SQL databases. A relational database like PostgreSQL or MySQL would be appropriate for structured data, including user profiles and scores.

4. **Memory Management**: The event-driven, non-blocking I/O architecture used by the Node.js platform is good at using memory and good for managing several connections at once with no overhead.

5. **Distributed Systems and Networks**: For "Draw It or Lose It," a microservices architecture might be used to help with communication between different platforms. Because each service can be updated and scaled individually, this architecture helps manage dependencies. Implementing circuit breakers and service discovery will ensure that the system can withstand partial outages and maintain a high degree of service availability in the event of network connectivity problems.
6. **Security**: To protect user information across various platforms, implement several security measures: Data security, Data Storage Security, Regular updates, and hotfixes when needed and using the best practices when coding.