Draw It or Lose It

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	07/21/24	Natalie Frye	This document will serve as a guide for the development of the game.

Executive Summary

This software document outlines the design approach for a web based distributed game application. The objective of this document is to provide a clear understanding of the software design problem and present a comprehensive solution to meet the client's requirements. The Gaming Room's decision to expand their game, Draw It or Lose It, to a web based platform brings new opportunities for increased accessibility and player engagement across multiple devices. Our goal is to transition the game from its current Android app format to a web based application, while still retaining its core elements and enhancing multiplayer functionality. To achieve this, the software design will incorporate modern web technologies, such as JavaScript, along with components to manage game logic and data storage. The design will emphasize performance and usability to deliver a high quality gaming experience.

Design Constraints

Developing the game application in a web based distributed environment introduces several design constraints that need to be considered:

1. **Network Latency:** The reliance on network communication in a web based environment introduces potential delays in data transmission. This may impact real time gameplay interactions, requiring optimization techniques such as predictive rendering and server side processing.

2. **Browser Compatibility:** Ensuring compatibility across various web browsers and devices adds complexity to development. The application must be thoroughly tested and optimized for different browser environments to guarantee a consistent user experience.

3. **Security Concerns:** The web-based nature of the application exposes it to security threats such as data breaches. Implementing security measures, including encryption protocols and secure authentication, is imperative to safeguard user data and prevent unauthorized access.

4. **Scalability:** As the game attracts more players and teams, the system must be able to scale effectively to accommodate increased traffic and demand. This requires a scalable design with efficient resource allocation and load balancing capabilities.

5. **Data Persistence:** Ensuring data integrity across distributed systems poses challenges in scenarios such as game state management and user progress tracking. Implementing reliable data storage solutions is essential to maintain a consistent gameplay experience.

Domain Model

The UML class diagram represents the Domain Model for the game application. It provides an overview of the main classes and their relationships, showcasing the key entities and behaviors of the system. The following classes are depicted in the diagram:

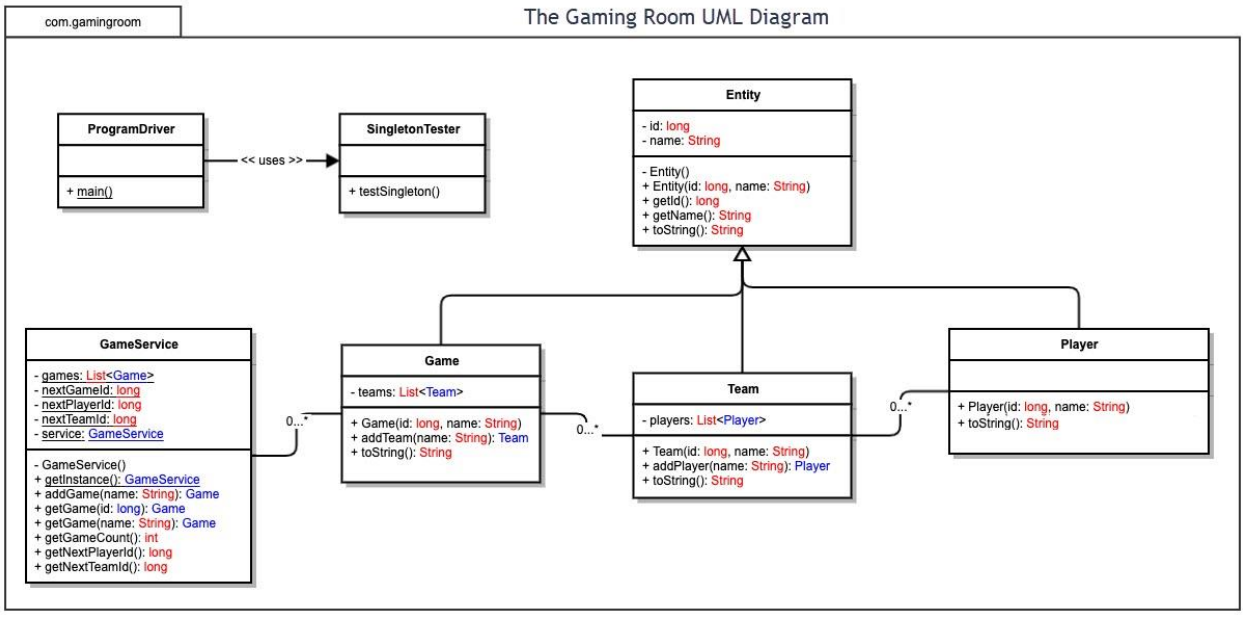
1. **Entity:** This is a base class that holds common attributes and behaviors shared by other classes in the domain model. It demonstrates the object oriented programming principle of inheritance, allowing other classes to inherit its properties and methods.

2. **User:** This class represents a user of the game application. It shows user specific information, such as username, password, and game statistics. The User class demonstrates the encapsulation principle by encapsulating data and providing methods to access and manipulate it securely.

3. **Game:** This class represents an instance of a game within the application. It contains information about the game's state, players, and game specific data. The Game class demonstrates the aggregation principle by aggregating multiple Player objects.

4. **Player:** This class represents a player participating in a game. It holds player specific information, such as score and current game status. The Player class demonstrates the association principle, showing the relationship between players and games.

5. **Leaderboard:** This class represents the leaderboard of top players in the game application. It tracks and displays



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	**Characteristics**: Mac can be used as a server for web based software applications, but it's less commonly chosen compared to Linux or Windows. It offers a Unix based environment, which is favorable for hosting web applications. **Advantages**: Unix based system provides security and stability. Mac Server includes features like Apache web server. **Weaknesses**: There are limited hardware choices for server grade Macs. It may be more expensive compared to Linux-based solutions.	**Characteristics**: Linux is a popular choice for hosting web based software applications. It offers a wide range of distributions suitable for server use, with stability and scalability. **Advantages**: Cost effective, highly customizable, and offers a wide range of server software options. Linux is known for its security and reliability. **Weaknesses**: It may require more technical expertise to set up and manage compared to Mac or Windows.	**Characteristics**: Windows Server is a commonly used platform for hosting web applications, especially for applications built using Microsoft technologies. **Advantages**: Excellent integration with Microsoft tools and technologies. Offers a range of web server options. It is a good support for .NET applications. **Weaknesses**: Licensing costs can be high. May not be as well suited for open source software development.	**Characteristics**: Mobile devices don't typically host web applications. Instead, servers are used to serve mobile app data. Any of the other platforms (Mac, Linux, Windows) can be used for mobile app backends. **Advantages:** Proximity to data for reduced latency. It provides cost efficiency for specific use cases. There is relatively easy deployment for small scale applications. **Weaknesses**: There are limited resources which can lead to performance issues. You can also run into reliability issues like battery drain and hardware failures. There is potential security risks due to vulnerabilities.

Client Side	**Software Development Considerations:** Developing for Mac clients typically involves using Apple's development tools such as Xcode. Costs can be moderate, and knowledge of certain languages may be required.	**Software Development Considerations:** Developing for Linux clients can vary depending on the distribution and desktop environment. Costs are typically low, but knowledge may be required for specific considerations.	**Software Development Considerations:** Developing for Windows clients often involves using Visual Studio. Costs can vary, but knowledge in .NET languages may be required.	**Software Development Considerations:** Developing for mobile devices involves platform specific development. Costs can vary depending on the number of platforms targeted.
Development Tools	**Relevant Programming Languages and Tools:** Xcode is the primary integrated development environment for Mac applications. It supports languages such as C++.	**Relevant Programming Languages and Tools:** Linux supports a wide range of programming languages. This can include Visual Studio Code and Eclipse.	**Relevant Programming Languages and Tools:** Visual Studio is the primary IDE for Windows development, supporting languages like C++ and more.	**Relevant Programming Languages and Tools:** For iOS, Xcode can be used. For Android, Android Studio and Java are common. Cross platform tools can streamline development for both platforms.

Recommendations

1. **Operating Platform**: The first recommendation is to consider the operating platform for the game. Since the game will run on several platforms, it is important to choose an operating platform that supports cross platform development. For example, web based games can use HTML, or JavaScript, which can run on multiple platforms including Windows, MacOS, Linux, Android, and iOS. Additionally, choosing a platform with a large development community can be beneficial as it allows for collaboration, knowledge sharing, and access to multiple resources.

2. **Operating Systems Architectures**: For cross platform development, it is recommended to choose an architecture that supports code reuse. The most common architecture for web based game development is client server architecture. Client server architecture is suitable for games that involve real time interaction. Or using event based architecture is suitable for games that require high performance computing and efficient memory management. Therefore, it is important to choose an architecture that matches the requirements and goals of the game.

3. **Storage Management**: Since the game will be web based, it is important to store game data in a cloud based environment to ensure easy access and distribution across different platforms. It is also important to choose a reliable, secure, and scalable storage management system to accommodate the growing number of users. Memory management is important to ensure the game runs smoothly and without errors. Choosing an architecture that supports efficient memory management can help reduce memory leaks and other performance issues that can affect game quality.

4. **Memory Management**: Games should be designed to optimize memory usage, especially for mobile devices. This can be achieved by reducing the use of large images and videos and using efficient encoding methods. Additionally, games should be designed to cache frequently accessed data to reduce the load times and improve overall performance.

5. **Distributed Systems and Networks**: Games will be accessed from different platforms and devices, and it is necessary to have a distributed system that can manage game traffic and ensure a smooth connection between different users. Choosing an architecture that supports load balancing and auto scaling can ensure that a game can handle many users and the game runs smoothly.

6. **Security**: It is important to ensure that the game is protected from cyber attacks, data breaches and other security threats. Choosing secure coding practices, data encryption, and user authentication can ensure game security and user data protection. In addition, it is recommended to conduct regular security checks and tests to detect and fix possible vulnerabilities in the game architecture.