Yanika Francis

7-1 Project three

The Game Room

CS – 230

Prof. Hecker

April 16, 2025

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	03/22/25	Yanika Francis	Initial 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 client, The Gaming Room, is looking to create a web-based game inspired by the 1980s TV show "Draw It or Lose It." They want the game to have multiple teams, each with different players, unique team names, and game names. The game cannot have any teams that are empty or have the same teams playing at the same time. To meet these goals, we will build a web-based system that uses cloud hosting for easy scaling. This will allow the game to work well on Windows, Mac, Linux, and mobile devices by using frameworks that support multiple platforms. We will also use data encryption and user authentication to protect sensitive information. This style will provide a smooth, secure, and scalable gaming experience, meeting the client's needs while ensuring the game is always available and performs well.

Requirements

Business Requirements:

- The game needs to allow **multiple players** to interact with each other in real-time.
- The platform needs to be **scalable** so it can handle more and more users as we grow in this web-based game.
- It should provide a **steady user experience** on both desktop and mobile devices.
- **Keeping your personal info safe** is super important, so making sure your login and security stuff works well is key. Protecting personal data plays an important part.
- The software should be a **reasonable cost**, using smart hosting and storage options.

Technical Requirements:

- The game needs to be **online/web based** and operate in a system where multiple computers work together.
- It should allow players to talk to each other in **real-time**.
- The app needs to be created using cross-platform tools like **HTML5, JavaScript, React, and WebSockets**.

- To be efficient, using **cloud-based storage** solutions.
- The backend should be developed using **scalable frameworks** like **Node.js** or **Django**.

Design Constraints

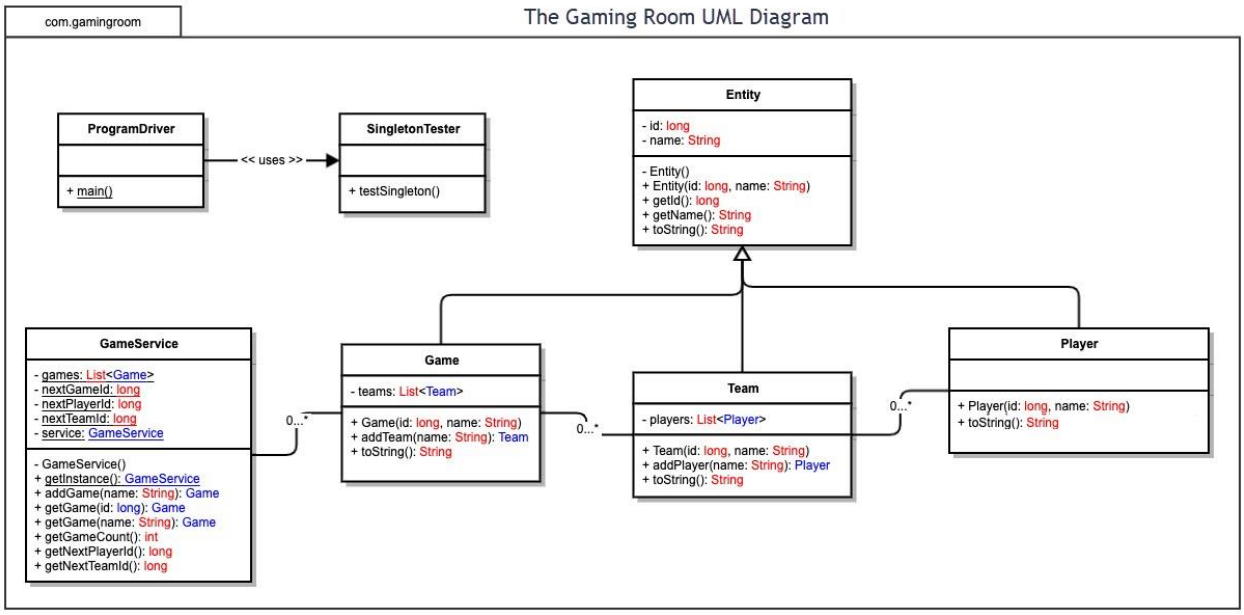
- **Web-Based Environment**: Since the application is created in a web-based setting, the game must be designed to work smoothly on all popular browsers and platforms. Using these ***Windows, Mac, Linux, and mobile devices***. This would require flexible development framework.
- **Network latency**: Being that the game is multiplayer, it needs a stable network for real-time updates.
- **Security**: Since we might have random groups of players of different sizes, implementing security measures from the beginning makes it easier to adjust and improve things later on. If we add security features later in the development process, we could face unexpected performance issues.
- **Scalability**: Since this game connects through the internet and can support big teams and many games, the system needs to handle a lot of users at the same time. To do this, it needs good database management and improvements on the server side. This could involve shifting some simple tasks to the user's device.
- **Storage and performance**: To improve file management and guarantee quick access to game assets, it is important to use cloud-based storage.

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

The UML class diagram defines the key relationships between objects in the game.



ProgramDriver: This class is the beginning of the program because it includes the main() function.

SingletonTester : This class is the test for the Singleton in GameService.

GameService : This class handles various game operations. It oversees games, teams, and players, following a Singleton pattern to ensure that only one GameService is created. It has connections with each related class, forming a chain that allows for zero to many relationships from Game to Team to Player.

Game: This class manages a game and contains the list of teams with the ability to manipulate teams. It has a zero to many relationships with Team.

Team: similar like a game, but it focuses on storing and changing players. Just like in a game, it can have zero to many connections with players.

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	macOS is reliable for running a web-based application, but it has some drawbacks. It offers less support for cloud-native features when compared to stronger server systems like Linux or Windows with Azure. This can make it harder to scale and deploy in the cloud, especially in setups that are spread out.	Linux distributions are recognized for their high security and stability, making them a great choice for hosting scalable web applications. They work well with cloud-native tools, and because Linux is open-source, there are usually no licensing fees for the operating system. This helps keep costs low for clients.	Windows provides different server setups and allows for flexible solutions using Microsoft Azure. Azure has many cloud-based choices designed to fit various project requirements. While Windows Server licenses may come with extra expenses, connecting with Microsoft's ecosystem can be advantageous for businesses.	Mobile platforms such as Android and iOS aren't made for hosting servers. They are designed for client-side use and lack the capability to handle large, cloud-based server setups. A solid mobile interface is crucial for the game's performance and to keep users engaged.
Client Side	macOS is great for developing apps designed for Apple's ecosystem, especially for iOS and macOS. However, this means that development is limited to the Mac environment, which isn't the best for making apps that work on multiple platforms. For projects like Draw It or Lose It that need to function on different systems, using macOS might need extra tools or adjustments.	Linux is highly adaptable and comes with support for many programming languages and frameworks straight away. This feature makes it an excellent choice for web development. However, it doesn't support some proprietary software, and using certain applications might need additional compatibility layers, which can complicate testing and deployment.	Windows platforms usually work well with web browsers, which makes them great for running web-based applications. They are often used for simple web browser support and are compatible with many desktop users. This makes Windows a reliable and easy option for client-side deployment.	Mobile devices play a key role in Draw It or Lose It, as the first version was designed for Android. Since the client aims to grow, it's essential to make sure the game works well on both Android and iOS. Mobile platforms are excellent for testing and using lightweight, responsive web apps, and they offer a simple way for users to engage with the game.

Development Tools	People use Xcode for creating native apps for iOS and macOS, while Visual Studio Code is popular for general development tasks. React is frequently used to create web interfaces. These tools are effective and easy to use, but developers should be knowledgeable about the Apple development environment.	Development tools for Linux are VS Code, Docker, and the Terminal. These tools are open-source and are commonly used by developers. Linux offers a solid DevOps environment, which is very helpful for distributed systems. However, developers might need to have some experience using command-line tools.	Common tools used for Windows development are Visual Studio, the .NET framework, and React for creating front-end applications. These tools are powerful and made for developing both desktop and web apps. However, some Microsoft products may need extra budget planning for their licenses.	In mobile development, the using tools are Android Studio for creating Android apps, Xcode for developing iOS apps, and React Native for making apps that work on both platforms. These tools help developers build apps that respond well and share code, but to work on both types of apps, you need to understand both systems.

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**: Run the game on cloud servers that use Linux, such as AWS, Google Cloud, or Azure. This will allow players on Windows, macOS, and mobile devices to join in and play more easily.
2. **Operating Systems Architectures**: Use a three-level system: the client side for the game interface, the server side for game logic, and the database for storage. Use cloud-based architecture to manage growth and maintain stability.
3. **Storage Management**: MySQL is a widely used and flexible data handling language in a database. MySQL supports multiple sizes and is used in most application domains. You can store game-related data in MongoDB Atlas or Firebase for flexibility and fast retrieval.
   Google Cloud Storage works great for storage management also.
4. **Memory Management**: Utilize garbage collection in Node.js to ensure efficient memory usage.
5. **Distributed Systems and Networks**: WebSocket communication is very powerful for platform-to-platform communication as a distributed system. This enables data exchange in real time regardless of platform. Redundancy and failover to avoid or largely minimize connection faults ought to be added as well.
6. **Security**: Keep logins safe by using OAuth 2.0 or Firebase Authentication. Encrypt data using SSL/TLS, protect APIs with DDoS defense, and make sure to enforce access control. Implement SELinux for strong security and regularly update systems to avoid weaknesses.