Draw It or Lose It

CS 230 Project Software Design Template

Version 3.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
3.0	02/16/2026	Esteban Garcia	Made final recommendations to client on which operating system to use for the game application, as well as detailed information on how the recommended operating system deals with game application security, concurrency, and storage.

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

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

Creative Technology Solutions has taken on a new client by the name of The Gaming Room who want to scale their Android-only game, Draw It or Lose It to multiple platforms. The main software design problem is going from a singular mobile platform to multiple other platforms like Linux, Mac, and Apple. The game will need to support many users from different operating systems logging in at the same time and having access to the same game as any other platform.

For the solution CTS recommends using a single GameService class to control the game. This service will manage the games, teams, and players while assigning the game sessions unique identifiers. Each game can include multiple teams and each team can have multiple players. This singular and central design keeps all the relevant game data secure and centralized in one location and allows the game to support multiple operating systems without needing to change the central game logic.

Requirements

< Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client's business and technical requirements in a clear and concise manner.>*

Design Constraints

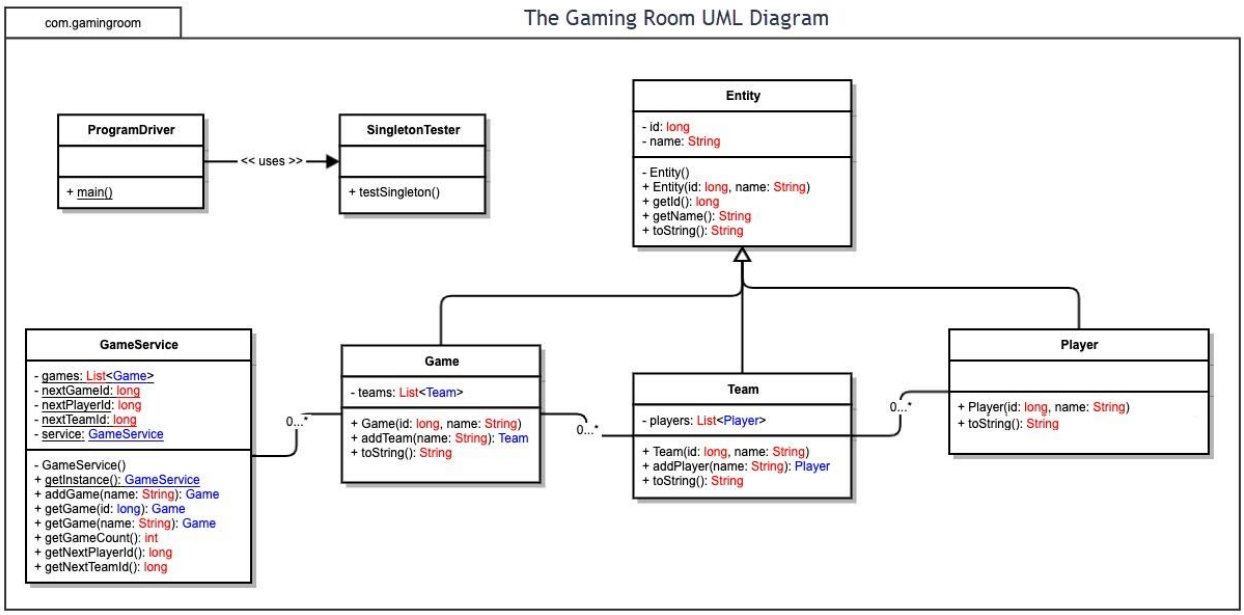
There exist various design constraints when developing the game application in a web-based distributed environment. One constraint in a network environment is players suffering high latency and dropped network connections. This means that the rounds in the game need to be timed on the server and not the user devices to prevent inconsistencies in game. The game must also manage concurrency well. It must provide a system of checks to verify that two games can't have the same name, players, and any other characteristics that shouldn't be repeated across instances. The game must also establish security and proper input validation. In a web-based game, as with any other web-based application, there needs to be error handling in place to prevent malicious users from injecting malicious code in place of a name on the game. Whether that be a player's name, a game's name or any other input in the game. Another important constraint is the difference in UI across many different platforms. To solve this, the UI needs to not be tied to one specific platform but rather be responsive and flexible to change across platforms.

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 Diagram below shows the main class structure of the existing game application. The main class is called the GameService manages the game and controls game creation. It calls the Game class with the method addGame to start a new game. It is from this central class that the classes Game, Team, and Player are derived from. The Game class represents a game session and contains a list of teams. The Team class calls the Player class and manages the list of players in a team. The Player class stores the player name. Altogether the Game, Team, and Player classes are child classes of the parent class Entity. The Entity class serves to hold common class attributes and names and clearly shows the object-oriented principle of inheritance. Encapsulation is used in the diagram by showing what data is private and can be accessed through getter methods.



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	Deploying a web app on a macOS server uses built-in Unix tools and third-party servers such as Apache. It can be stable for small scale apps but using it for large-scale deployments comes with some drawbacks. The cost to deploy an app on macOS comes with costly Apple licensing costs, making it not sustainable for scalable hosting.	Using Linux to deploy a web app is probably the best option for both small- and large-scale web app deployments and for companies who want to keep server costs low. Linux offers stellar performance, scalability, and most Linux distributions are open source and come with little to no licensing costs.	Windows has excellent support and reliability since it is widely used across many industries and companies. However, it comes with higher licensing costs than Linux. It can be an excellent choice for companies who can pay the licensing costs long term and want Windows integration.	Using a mobile device to deploy a web app on the server side would be unfeasible. The platform lacks the hardware, scalability, and reliability to host a web app.

Client Side	Supporting Mac clients requires ensuring it is compatible with web browsers like Safari and the macOS versions of Chrome or Firefox. It also needs to ensure that there is the capability to integrate features specific to macOS on the client side. Development costs are moderate as it is widely used and supported by developers and designers. Development needs to focus mainly on ensuring combability is supported across browsers, specifically Safari.	Using Linux to support clients requires developing cross-platform support across many Linux distributions. As such, it also requires knowledge of developing client-side applications for these distributions. Licensing costs are usually relatively low compared to macOS and Windows.	Windows clients make up the largest desktop user base. Developing an app for the Windows client requires familiarity with Windows development tools as well as the different Windows OS versions that are still supported. Licensing costs are often moderate.	Developing an app for mobile operating systems requires having support for the different screen sizes, and operating systems on mobile. Development time and cost are often higher than desktop due to supporting the many different devices and screen sizes on mobile. Development expertise is needed across many different mobile platforms.

Development Tools	HTML, CSS, and JavaScript are core technologies needed to support web application development across all the existing platforms. Other technologies for macOS include frameworks such as Spring Boot and Java or Node.js. VS Code and Xcode are common IDEs used for development. Xcode specifically is needed to test applications on Safari.	Linux uses the same three core languages to develop a web application. In addition, popular tools are Java, Eclipse, IntelliJ, and VSCode. Most of these tools are open-source and have minimal licensing costs.	Development in Windows usually uses the Windows-specific framework of .NET. Along with Java, HTML, JavaScript, and CSS. The main two IDEs are Visual Studio and VSCode, which are developed by Windows. Of course there are also third-party IDEs that can be used. Developers may need Windows expertise to code on this platform.	Building a web app on mobile requires languages such as Java, HTML, CSS, and JavaScript along with responsive frameworks like Angular.js and React. Tools used are Android Studio and Xcode for iOS. Coding for an iOS platform requires the use of macOS for testing which may add to the development time and cost.

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**:

I recommend Linux as the operating platform to host the server that will allow The Gaming Room to expand their game to many different client environments. The reason for this is that Linux allows for stable, scalable, and universal support for web hosting. Using Linux to host your application will also reduce high licensing costs that you would see with Windows or MacOS. Since Linux works across many different client platforms, it makes using this OS the best fit since developers don't need to rewrite server-side code when adding new client platforms.

2. **Operating Systems Architectures**:

A server running on the Linux OS has various operating system architectures. The Linux kernel is used to manage operations such as memory allocation, networking, and file input/output. Applications interact with hardware safely using the kernel and system calls. Linux also supports multithreading, meaning the game application will be able to support many user and game sessions occurring at the same time. For networking, Linux uses standard TCP/IP networking protocols to secure web traffic, as well as supports load balancing and reverse proxies. These protocols keep game traffic safe and user data secure. These built-in architectures support reliable deployment and use of the game on client platforms.

3. **Storage Management**:

An appropriate storage management system to use with the Linux operating system is using a relational database system such as PostgreSQL alongside the Linux distribution file system for static data. While there exist other database systems such as MySQL, using PostgreSQL allows for better scalability and long-term efficiency as the game grows in users. Using a PostgreSQL database, the application can store game data such as user accounts, game sessions, individual player records, and high scores, and any other data applicable. Using this relational database, certain rules can also be enforced, such as each player and game having a unique ID. When dealing with static data, such as the 200 high-quality images the game will use for the puzzles, storing them in the database would be inefficient. Rather, they should be stored on the cloud or the server file system, while the database stores paths to the images.

4. **Memory Management**:

The Linux operating system uses several memory management techniques suitable for the Draw It or Lose It game. First, when implementing the server using Java, the Java Virtual Machine uses garbage collection strategies that are implemented to reduce memory use and improve performance. Another memory management technique Linux uses well is efficient and reliable caching. The Linux system will use free memory to cache resources that are frequently accessed during the game session, such as game assets, user configurations, and other data. Using the appropriate memory management techniques for the game application will allow for optimized performance and runtimes, leading to greater player satisfaction.

5. **Distributed Systems and Networks**:

To accomplish the game to work throughout many various platforms, we can use distributed software and the network that connects these devices. To support these various platforms, the game should be designed as a distributed client-server system. In this system, one Game Service serves as the central server. It then distributes data such as game sessions, user sessions, scoring, and rules to various client platforms. This enhances game security and ensures that there are no timing issues between platforms. To ensure that gameplay users have the best performance, the protocol WebSockets can be used. WebSockets is one of the best protocols for gaming applications because it offers low-latency, and consistent, and stable communication between the server and its clients after the connection is first established.

As the number of players grows, the system can scale to ensure performance is maintained. Using load balancers, the application can distribute game traffic across as many server instances as needed. Horizontal scaling, which is when distributing application traffic across many nodes or machines, can also be used. This will allow many game sessions to run concurrently with necessary game data preserved across the sessions.

To account for outages and disruptions during gameplay, the server should prioritize keeping the game session running for those clients still connected to the game instance. It should implement retry logic to allow a disconnected player an opportunity to regain access to the game instance. As outages and player disruptions can occur, it is important that all time-sensitive data, such as game round timers, and user scoring during rounds, are controlled by the server, not the clients.

6. **Security**:

To ensure game and user security is protected, Linux uses strong permissions, firewalls, and support for secure deployment protocols. To secure web traffic, HTTPS and WebSockets should be used. These two protocols protect data during transit from unauthorized interception. Using built-in Linux firewall tools, the server can prevent any ports from being used except for necessary ones such as HTTPS. Using secure tokens to authenticate users and game sessions will prevent spoofing game scores and timers during game rounds. On the server side, any client input, such as game names, player names, etc., must be validated to ensure the input is not an attempt to inject malicious code to the server. Proper error handling should also be implemented on the server to prevent sensitive user and game data from being exposed to wrongdoers.

Properly securing user data on the database or server should also be a priority. Using hashing to store passwords instead of storing them as plaintext will ensure user passwords aren't accessed. Unusual traffic spikes should also be monitored to detect Denial-of-Service attacks that can disrupt game sessions and cause frustration among players. Linux can provide these traffic and logging tools to secure the game against malicious traffic.