The Gaming Room

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/24/2024	Brianna Peoples	Updated the summary, design constraints, and domain model.

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 is expanding its hit game, Draw It or Lose It, to a web-based system. This expansion aims to attract a larger audience and ensure it is accessible to its players anytime and anywhere. My goal is to design a system that ensures the game runs smoothly and works on all devices while keeping everything secure.

Requirements

Business Requirements:
- Expand Draw It or Lose It to support web-based access.
- Ensure the game can handle more users as it grows.
- Maintain high availability and minimal downtime.

Technical Requirements:
- Support multiple operating platforms, including Linux, Mac, Windows, and mobile devices.
- Provide secure communication and data management.

Design Constraints

Cross-Platform Compatibility
- Constraint: The game must function seamlessly across multiple platforms, including mobile devices.
- Implications: Need to implement APIs to enable communication between different devices, using a system that manages all interactions through a main server.

Scalability
- Constraint: The system must handle increased users and growing data.
- Implications: Need to use a cloud-based system and load balancers to allow the system to efficiently scale.

Security
- Constraint: User data must be protected across platforms.
- Implications: Need strong encryption and secure multi-factor authentication protocols.

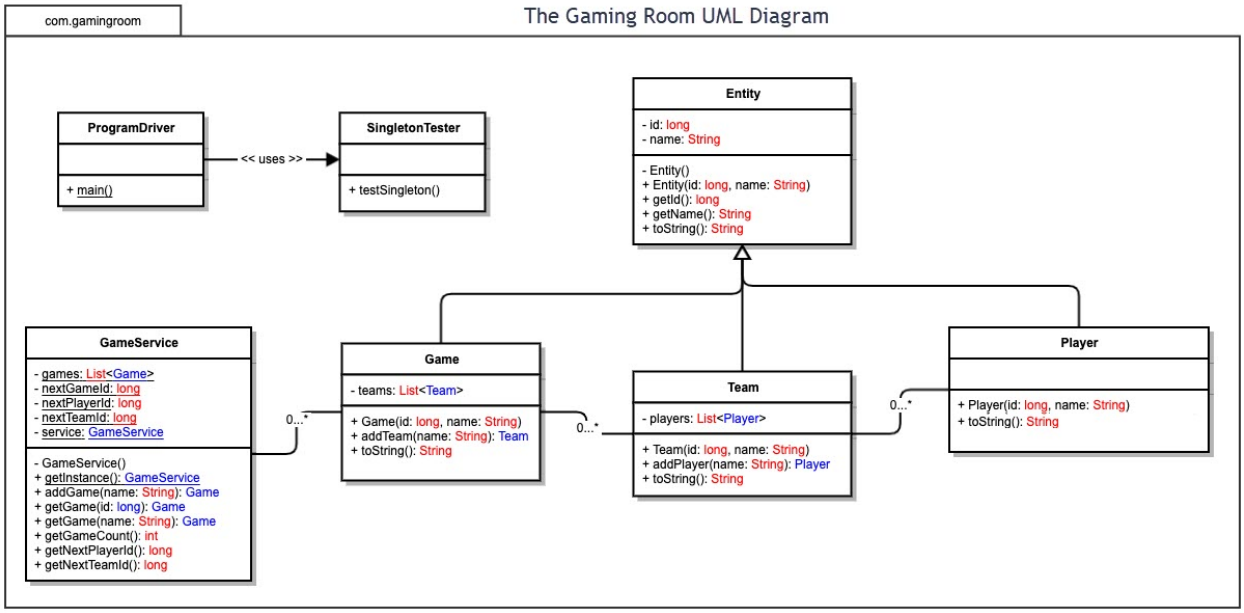
System Architecture View

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 for The Gaming Room's application shows how different parts of the game system all work together. The entity class is the base for the other main parts of the system which include the game, team, and player. The entity class includes shared features like an id and name, this helps to avoid repeating the same code in different places. The game service class allows us to manage everything, such as adding games, teams, and players, and also ensures the system works smoothly. The game class

holds the list of teams, and each team has a list of players. This shows how everything is connected. This setup makes it easy to manage the relationships between games, teams, and players. The program driver class starts the program, and the singleton tester class checks that the singleton design is working correctly. The program uses a singleton design, meaning only one version of the game service runs to keep everything organized. This design also uses basic object-oriented programming principles like inheritance and encapsulation. This setup makes the system efficient, easy to expand, and well-organized to meet the needs of the game.



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 works well for small to medium web hosting, but it's not usually used for big setups because it's less customizable and costs more. Licensing costs for macOS servers are included with the hardware purchase allowing there to be no additional server OS fees. While macOS is simple and secure, its limited use for servers is a downside.	Linux is a popular choice for hosting web-based applications because it is flexible, reliable, and cost-effective. Many versions of Linux are free to use and have no extra licensing costs. Although it can be tricky to set up and keep secure, It works well with tools for running servers and can handle a lot of users.	Windows Server is a great option for hosting web applications. It's easy for administrators to use and works well with Microsoft-based systems. Licensing can be expensive and it also may not handle growth as well as Linux. It also uses more system resources.	Mobile platforms don't directly host server applications. They use APIs or back-end servers, like Linux or Windows, to handle requests, process data, and send the content to the app. Since the mobile platforms mainly allow for users to interact with the app, they don't deal directly with server-side setup.
Client Side	Developing an app for macOS clients requires ensuring compatibility with Safari and other browsers like Chrome and Firefox. To do this, we can use tools like React or Vue.js to make the app work across different platforms smoothly. Time and cost considerations are moderate, but expertise in macOS development is needed for creating specialized features.	Linux desktops have to make sure the web app works well on browsers like Chrome and Firefox. Development costs are low because of its compatibility with modern frameworks. However, making sure the app runs smoothly on different Linux versions can be hard without a lot of testing.	Windows users primarily rely on browsers like Chrome or Firefox. Making sure its compatible with these platforms is pretty straightforward when using HTML5 and JavaScript frameworks. Development time and expertise are similar to macOS but may require extra testing due to a larger range of hardware setups.	Developing for Android and iOS requires ensuring that the web app is responsive and works on smaller screens. Frameworks like React Native or Flutter make the development process easier. However, additional costs and time are requiredfor improving features and making sure the app functions properly across various mobile platforms.
Development Tools	MacOS uses tools like Xcode, Visual Studio Code, or IntelliJ IDEA. These tools support development with languages like Swift or JavaScript.	Linux supports free/open-source IDEs like Visual Studio Code or Eclipse. Languages like JavaScript, Python, and Java are most commonly used.	Windows development tools include Visual Studio or Visual Studio Code. It works well for .NET development and cross-platform tools.	For mobile development, tools like Android Studio, Xcode, React Native, and Flutter are most common. These tools make code-sharing easier but may need extra expertise for full use.

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**: For the operating platform I would use a cloud-based service like Amazon Web Services or Microsoft Azure. These platforms make it easy to scale the app and allow it to work across different devices and systems as well. Using tools like Docker and Kubernetes will ensure the app runs smoothly on any platform whether it's Windows, macOS, Android, or iOS.

2. **Operating Systems Architectures**: The server platform should run on Linux because it's reliable, fast, and widely used for hosting apps. Linux is also free and can be customized for the app's needs. On the user side, we can use tools like Flutter or React Native to make sure the app works well on smartphones, tablets, and computers.

3. **Storage Management**: To store data, Azure Blob Storage would be a great option. This system is fast, reliable, and can handle a lot of data without any issues. For user information and game data, a database like PostgreSQL is a good choice, and for quick access to temporary data we can use Redis or Memcached, both which are very fast.

4. **Memory Management**: The way Linux handles memory is also a good fit for the app. It only loads what's needed into memory, uses extra disk space if memory gets full, and keeps frequently used data in memory to make things faster. These techniques will help the app run well, even when lots of people are playing the game at the same time.

5. **Distributed Systems and Networks**: To make the app work on different platforms, we can break it into smaller parts that handle specific tasks, like game logic or chat features. These parts can talk to each other using APIs. For real-time gameplay, we can use WebSockets to make the app feel faster and more responsive. To make sure players have a good experience no matter where they are, we can use a Content Delivery Network to quickly load images, sounds, and videos.

6. **Security**: Security is extreamly important for protecting user information. To keep data safe, we can use encryption when data is sent between devices and the server. User accounts will be protected with secure login systems, and sensitive information like passwords will be encrypted in the database. The cloud platform will also help by providing tools to control access, monitor for suspicious activity, and respond to potential threats.