Module Seven –

CS 230 Project Three

KEYNEISHA D. MCNEALEY

PROFESSOR HECKER

08/08/2024

The Game Room

CS 230 Project Software Design – Project Three

Version 2 .0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	07/13/2024	Keyneisha Davion Mcnealey	Initial Submission of software design CS 230 Project One
2.0	08/08/2024	Keyneisha Davion Mcnealey	Updates on system architectures

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 software design problem presented is to develop a web-based game application, "Draw It or Lose It," in a distributed environment. The solution proposed is to design a system that can be deployed on multiple operating platforms, including Mac, Linux, Windows, and mobile devices. This document outlines the design constraints, system architecture, domain model, evaluation, and recommendations for the development of the game application.

Requirements

The client's business and technical requirements include:

- Developing a web-based game application, "Draw It or Lose It"
- Deploying the application on multiple operating platforms, including Mac, Linux, Windows, and mobile devices.
- Ensuring the application can communicate between various platforms
- Protecting user information on and between various platforms.

Design Constraints

The design constraints for developing the game application in a web-based distributed environment are as follows:

Scalability: The application must accommodate a large number of users and games simultaneously, ensuring the system can scale both horizontally and vertically to meet increasing demands.

Security: The application must safeguard the security and integrity of user data, including game state, player information, and authentication credentials.

Compatibility: The application must be compatible with multiple operating platforms, including Mac, Linux, Windows, and mobile devices, to ensure a seamless user experience across different environments.

Performance: The application must deliver fast and responsive gameplay, ensuring the system can handle complex game logic and rendering without compromising performance.

Reliability: The application must be reliable and fault-tolerant, ensuring the system can recover from errors and failures without affecting the user experience.

Maintainability: The application must be easy to maintain and update, ensuring the system can be modified and extended without compromising its integrity.

Implications on Application Development

These design constraints have significant implications for application development, including:

Choice of Programming Languages and Frameworks: The selection of programming languages and frameworks must be based on their ability to meet the scalability, security, and performance requirements of the application (Oracle, 2022).

Database Design: The database design must be optimized for scalability, security, and performance, ensuring the system can handle large amounts of data and user traffic.

System Architecture: The system architecture must be designed to ensure scalability, reliability, and maintainability, including the use of load balancers, caching mechanisms, and content delivery networks.

Testing and Quality Assurance: Thorough testing and quality assurance must be performed to ensure the application meets the design constraints and provides a seamless user experience.

Recommended Operating Platforms

Based on the design constraints, the following operating platforms are recommended for developing the game application:

Linux: Linux is a suitable choice for hosting the game application, offering scalability, security, and performance. Linux-based operating systems such as Ubuntu and CentOS are popular choices for web-based applications (Linux Foundation, 2022).

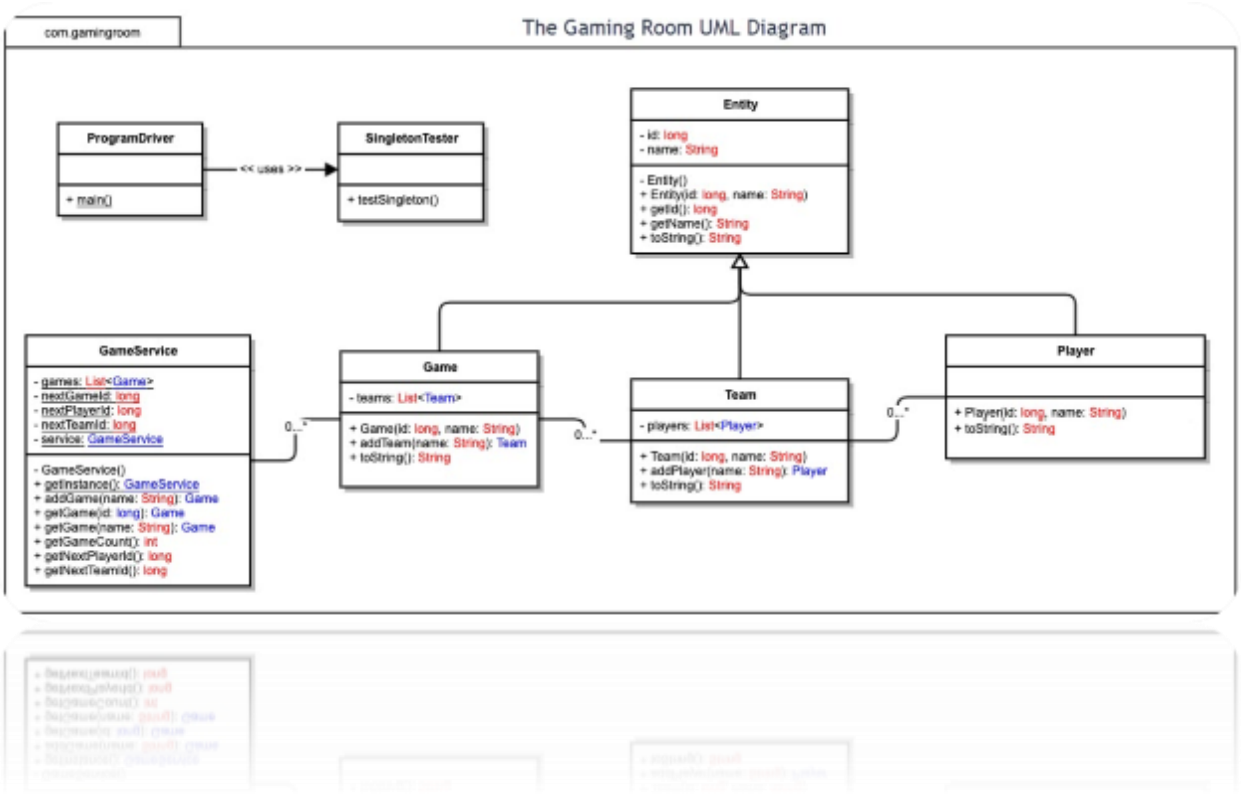
Windows: Windows is a widely used operating system that can provide a scalable and secure environment for the game application. Windows Server and Azure are popular choices for hosting web-based applications (Microsoft, 2022).

Cloud-Based Platforms: Cloud-based platforms such as Amazon Web Services (AWS) and Google Cloud Platform (GCP) offer scalable and secure environments for hosting web-based applications. These platforms provide a range of services, including compute, storage, and database management, that can be used to develop and deploy the game application.

System Architecture View

Not applicable

Domain Model



The UML class diagram presented comprises four distinct classes: Entity, Game, Team, and Player. These classes are interconnected through inheritance and composition. The Entity class functions as a foundational base class for the other three classes, offering a shared attribute, id, along with its corresponding getter method. The Game class maintains a one-to-many relationship with the Team class, while the Team class similarly holds a one-to-many relationship with the Player class. This architectural design exemplifies key object-oriented programming principles, including inheritance, encapsulation, and composition.

Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Mac is a suitable option	Linux is a popular choice	Windows is a widely used	Mobile devices are not suitable

	for hosting a web-based software application, offering a stable and secure environment. However, it may require additional configuration and setup.	for hosting web-based applications, offering flexibility, scalability, and cost-effectiveness.	operating system, offering a familiar environment for developers. However, it may require additional security measures.	for hosting a web-based software application, but can be used as clients to access the application.
Client Side	Developing a client-side application for Mac requires consideration of the operating system's specific requirements and compatibility issues.	Developing a client-side application for Linux requires consideration of the various distributions and compatibility issues.	Developing a client-side application for Windows requires consideration of the operating system's specific requirements and compatibility issues.	Developing a client-side application for mobile devices requires consideration of the device's operating system, screen size, and input methods.
Development Tools	Relevant programming languages and tools for Mac include Java, Swift, and Xcode.	Relevant programming languages and tools for Linux include Java, Python, and Eclipse.	Relevant programming languages and tools for Windows include Java, C#, and Visual Studio.	Relevant programming languages and tools for mobile devices include Java, Swift, and Android Studio.

Table 1. Development Requirements.

This table outlines the development requirements for various platforms, emphasizing the unique considerations and tools pertinent to each.

Recommendations

Based on the evaluation, I recommend the following:

Operating Platform

Recommendation: It is recommended to utilize a **Linux-based server platform**, specifically **Ubuntu Server**. Linux is renowned for its stability, security, and scalability, making it an ideal choice for expanding "Draw It or Lose It" to various computing environments.

Operating Systems Architectures

Description: The selected operating platform, Ubuntu Server, employs a **monolithic architecture**. In this architecture, the kernel manages all core functions, including process management, memory management, and device drivers. This design ensures efficient performance and robust security.

Storage Management

Recommendation: Implement the **Zettabyte File System (ZFS)** for storage management. ZFS provides high storage capacity, data integrity verification, and efficient data compression. It is well-suited for handling large volumes of game data and ensuring data reliability.

Memory Management

Explanation: Ubuntu Server utilizes several memory management techniques:

- **Paging:** Divides memory into fixed-size pages to manage virtual memory efficiently.
- **Swapping:** Moves inactive pages to disk to free up RAM for active processes.
- **Segmentation:** Divides memory into segments based on the type of data (code, stack, heap) to improve memory access speed.

Distributed Systems and Networks

Implementation: To enable communication between various platforms, a **microservices architecture** with **RESTful APIs** is recommended. This approach allows different components of "Draw It or Lose It" to interact over a network, ensuring scalability and fault tolerance. The network can be managed using **Kubernetes** for container orchestration, ensuring high availability and load balancing.

Security

Measures:

- **Data Encryption:** Use **SSL/TLS** for encrypting data in transit and **AES** for data at rest.
- **Authentication and Authorization:** Implement **OAuth 2.0** for secure user authentication and **role-based access control (RBAC)** for authorization.
- **Regular Updates:** Ensure the operating system and all software components are regularly updated to patch security vulnerabilities.

By addressing these areas, a robust and scalable software design for "Draw It or Lose It" can be created, meeting the client's requirements for expansion, performance, and security.

References:

Oracle. (2022). Java SE Documentation. Retrieved from https://docs.oracle.com/en/java/javase/11/docs/api/index.html

Linux Foundation. (2022). Linux Kernel Documentation. Retrieved from https://www.kernel.org/doc/

Microsoft. (2022). Windows Server Documentation. Retrieved from https://docs.microsoft.com/en-us/windows-server/