Draw It or Lose It
CS 230 Project Software by Brandon Canady
Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.2	02/22/25	Brandon Canady	Final Analyses

Executive Summary

The Gaming Room requires a distributed system that efficiently manages game entities—games, teams, and players—in a web-based environment. The solution must ensure name uniqueness, support scalability, and maintain high performance and security across different platforms..

- The client requires a distributed system that can efficiently manage games, teams, and players while maintaining unique identifiers and names. The system must support scalability and maintain strict rules for name uniqueness to prevent conflicts.

Critical Information:

- Highlight the importance of unique identifiers, immutability of certain fields, and the use of design patterns to ensure scalability and maintainability.

Requirements

- Entity Management: Create and manage games, teams, and players with unique names.
- Hierarchical Structure: Games contain teams; teams contain players.
- Performance: Handle concurrent requests in a distributed environment.
- Gameplay Rounds: Each game has four one-minute rounds; drawings complete by the 30-second mark with bonus guessing periods.
- Network Efficiency: Low latency and secure data synchronization are critical.

Design Constraints

- The application must operate over a network, requiring efficient data synchronization and minimal latency.

- Security considerations, such as preventing unauthorized access to game data, must be addressed.
- Names for games, teams, and players must be unique to prevent ambiguity and maintain data integrity.
- The design must allow the addition of new games, teams, and players without significant refactoring or performance degradation.
- Drawings must be rendered incrementally and completed by the 30-second mark, requiring precise timing mechanisms.
- Rounds must be enforced with a one-minute limit, including additional logic for 15-second bonus guessing periods.

System Architecture View

Domain Model

The UML class diagram represents the structure of the Gaming Room application. It consists of the following key components:

1. Entity Class:

- A base class containing common attributes id and name.
- Provides methods for retrieving the ID and name and a toString() method for representation.

2. Game Class:

- Inherits from Entity and includes a list of teams.
- Provides methods for adding a team and converting game information to a string.

3. Team Class:

- Inherits from Entity and includes a list of players.
- Provides methods for adding a player and converting team information to a string.

4. Player Class:

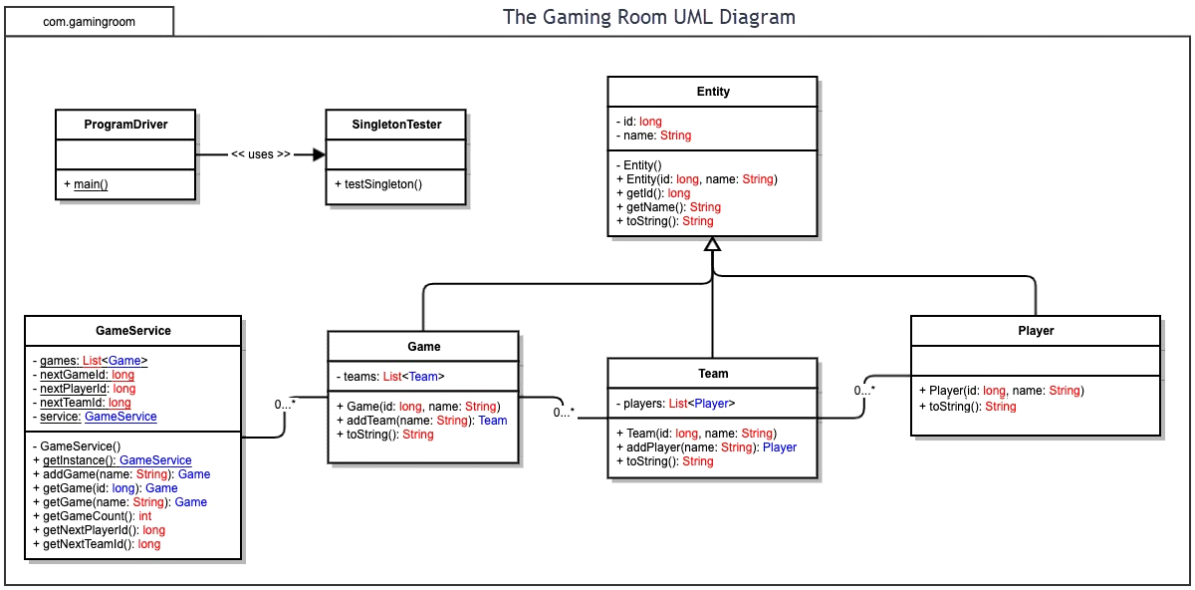
- Inherits from Entity and represents individual players.
- Provides a toString() method for representation.

5. GameService Class:

- Implements the Singleton pattern to ensure only one instance of the service exists.
- Manages the lists of games and provides methods for adding games, retrieving games by ID or name, and generating unique IDs.

5. ProgramDriver and SingletonTester Classes:

- Serve as entry points for testing and demonstrating the functionality of the application.



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 offers a robust development environment with native Unix compatibility and high security. However, it can be expensive and less common for production servers.	Linux is an industry-standard for hosting due to its flexibility, scalability, and cost-effectiveness . Weaknesses include a steeper learning curve for beginners.	Windows servers provide strong enterprise support and ease of use, but they can be costly and less customizable compared to Linux.	Hosting on mobile devices is impractical due to hardware limitations and restricted OS capabilities.
Client Side	Supporting Mac requires additional testing and considerations for macOS-specific behaviors, which may increase development time	Linux clients are less common, but supporting them involves ensuring compatibility with open-source libraries and tools.	Windows is widely used, so supporting it ensures broader reach. However, testing multiple versions of Windows may	Mobile development requires cross-platform frameworks, such as React Native or Flutter, to minimize duplication and

	and cost.		increase costs.	effort.
Development Tools	Development on Mac can leverage tools like Xcode, Visual Studio Code, and languages such as Swift, JavaScript, or Python.	Linux developers often use VS Code, IntelliJ IDEA, or Eclipse. Programming languages like Python, JavaScript, and Java are common.	Windows development tools include Visual Studio, VS Code, and support for languages like C#, Python, and JavaScript.	Mobile development can use tools like Android Studio, Xcode, or cross-platform frameworks like React Native or Flutter.

Recommendations

Operating Platform:

Recommendation:
Adopt a cloud-based server platform (e.g., AWS, Google Cloud, or Microsoft Azure). This solution provides the flexibility to expand Draw It or Lose It into other computing environments, enabling seamless integration with various client platforms (Mac, Linux, Windows, and mobile).

Rationale:

- Scalability: Cloud platforms offer dynamic resource allocation that grows with the application's demand.
- Cross-Platform Compatibility: Enables consistent performance across diverse operating systems.
- Robust Ecosystem: Integrated services for compute, storage, networking, and security.

Memory Management:

Recommendation:
Rely on the Linux-based cloud platform's advanced memory management techniques.
Details:

- Dynamic Memory Allocation: Linux allocates memory on demand, optimizing resource usage based on real-time load.
- Caching and Virtual Memory: Enhances performance by using caching strategies and virtual memory, reducing latency during peak loads.
- Container Resource Limits: Docker and Kubernetes allow precise control over memory usage per container, ensuring that each microservice has the resources it needs without impacting others.

Distributed Systems and Networks:

Recommendation:
Architect the application as a microservices-based distributed system that communicates via RESTful APIs or GraphQL, supported by a robust network infrastructure.
Details:

- Inter-Service Communication: Microservices interact over standardized APIs, ensuring loose coupling and ease of maintenance.
- Load Balancing & Auto-Scaling: Cloud-native load balancers distribute traffic evenly; auto-scaling mechanisms dynamically adjust resource allocation to handle varying loads.
- Fault Tolerance: Redundant services and failover strategies ensure that connectivity issues or outages in one component do not cripple the entire system.
- Network Dependencies: The distributed architecture accounts for potential connectivity challenges by implementing service meshes and monitoring tools to detect and mitigate outages promptly.

Security:

Recommendation:
Employ comprehensive security measures offered by the cloud platform to protect user information both on and between various platforms.
Details:

- Data Encryption: Utilize HTTPS for data in transit and encryption at rest to secure stored data.
- Authentication & Authorization: Implement secure methods such as OAuth 2.0, token-based authentication, and multi-factor authentication (MFA) to protect access to user accounts and services.
- Access Control: Use Identity and Access Management (IAM) policies to restrict access based on the principle of least privilege.
- Network Security: Configure firewalls, Virtual Private Clouds (VPCs), and intrusion detection systems (IDS) to safeguard the network perimeter.
- Regular Audits and Compliance: Leverage built-in cloud security audits and maintain compliance with data protection regulations to ensure continuous monitoring and remediation of vulnerabilities.

Final Conclusion

For Draw It or Lose It, a cloud-based, Linux-powered server architecture is the optimal solution. This approach provides:

- Scalability to expand into various computing environments.
- Robust memory and storage management to handle dynamic workloads and large datasets.
- Efficient distributed systems that support seamless communication across platforms.
- Comprehensive security measures to protect sensitive user information.
- By implementing these recommendations, The Gaming Room will achieve a highly responsive, secure, and scalable environment that meets current demands while positioning the application for future growth.