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	06/02/2024	Bryce Schultz	Added information and details.
2.0	06/16/2024	Bryce Schultz	Verified that document meets Project 2 requirements. I didn't realize when I originally turned this in that I wasn't supposed to do the comparisons.
3.0	6/22/2024	Bryce Schultz	Verified that document meets Project 3 requirements, made some refinements to the final section.

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, a game development company, wants to expand their game "Draw It or Lose It" to a web-based distributed environment, allowing for multiple teams to play at the same time. The game involves guessing what is being drawn based on images rendered from a library of stock drawings. The client requires a software design document and a functional prototype of the game application.

To address these requirements, I will be developing a web-based version of the game using Java. The application will follow object-oriented programming principles and utilize design patterns such as 'singleton' and 'iterator' to ensure that the games, teams, and players are unique. The application will be designed to support multiple platforms and allow for future expansion.

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

1. The application must be developed for a web-based distributed environment.
2. The game must support multiple teams and players simultaneously.
3. Game and team names must be unique.
4. Only one instance of the game can exist in the memory at any given time.
5. The application should be scalable and extensible to accommodate future requirements.

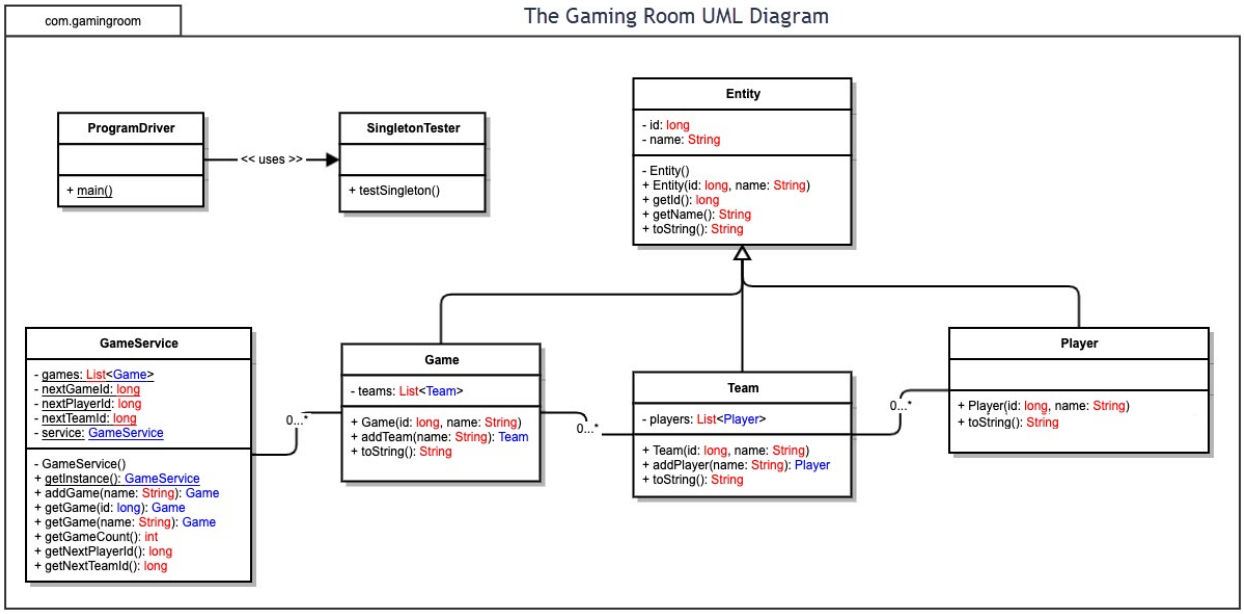
System Architecture View

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

Entity: An abstract base class that holds common attributes (id and name) and behaviors for all entities for the application.
Game: Represents a game instance and contains a list of teams. It inherits from the Entity class.
Team: Represents a team participating in a game and contains a list of players. It also inherits from the Entity class.
Player: Represents an individual player belonging to a team. It also inherits from the Entity class.
GameService: A singleton class responsible for managing game instances, generating unique identifiers, and providing utility methods for adding and retrieving games.

Relationships:
- A game has a composition relationship with Team, so a game consists of one or more teams.
- A team has a composition relationship with player, so a team consists of one or more players.
- The GameService class acts as a centralized manager for game instances and provides methods for adding and retrieving games.



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	- Stable and secure Unix-based OS - Easy to set up and maintain - Limited support for servers - Higher costs for hardware across the board	- Highly stable and secure. - Free and Open Sourced, and highly customizable. - Dominant in the server world.	- Wide adoption for enterprise systems - Extensive support and tooling - Integration with Microsoft's stack - Higher licensing costs	- Limited server-side capabilities - Rely on cloud-based services - Scalability and performance challenges that need addressed
Client Side	- Consistent user experience - Proprietary hardware and OS - Limited level of buy in from developers - Higher Development costs (fee to be Apple Developer)	- Varied user experience across distributions - Requires technical proficiency (depending on the distro) - Fragmented in terms of popularity	- Dominant desktop OS - Wide range of supported hardware - Extensive application ecosystem - Easier for end-users to adopt to	- Diverse range of devices and OSes - Touchscreen and mobile-optimized - Platform specific development - Varying screen sizes and device capabilities
Development Tools	- Xcode IDE for native apps - Swift and Objective-C languages - Cocoa frameworks - Well-documented APIs	- Wide range of open-source tools - GCC, Clang compilers - Java, Python, C/ C++ support - Extensive command line tools / TUI apps	- Visual studio IDE - .NET framework - C#, C++, Visual Basic languages - Extensive documentation and resources	- Android studio for Android apps - Xcode for iOS apps - Java and Kotlin for Android - Swift and Objective-C for iOS - Cross-platform tools like React Native and Flutter

Recommendations

Based on the analysis of various system architectures and the specific needs of The Gaming Room for their "Draw It or Lose It" game, we recommend the

1. Operating Platform

We recommend using Linux as the primary operating system for server-side deployment. Specifically, we suggest using Ubuntu Server, a popular and well-supported Linux distribution. This choice is based on the following factors:

- Open-source and cost-effective
- Highly stable and secure
- Excellent performance for web-based applications
- Wide range of supported hardware architectures
- Strong community support and regular updates

2. Operating Systems Architectures

Linux uses a modular and layered architecture, which consists of the following key components:

- Kernel: The core of the OS, managing system resources, process and memory management, and device drivers.
- Shell: The user interface for interacting with the OS through commands.
- Filesystem: A hierarchical structure for organizing and storing files and directories.
- Networking: Support for various networking protocols and tools for network configuration and management.
- Libraries and Utilities: Extensive collection of reusable code and tools for system and application development.

3. Storage Management

For storage management, we recommend using the ext4 (fourth extended filesystem) for the following reasons:

- Journaling support, ensuring data integrity and faster recovery after system crashes
- Backwards compatibility with ext2 and ext3
- Support for large file systems and large files
- Improved performance and reliability compared to its predecessors

4. Memory Management

Linux employs several memory management techniques that will benefit the "Draw It or Lose It" game:

- Virtual Memory: Provides each process with its own virtual address space, allowing multiple processes to run concurrently without interference.
- Paging: Divides memory into fixed-size pages, enabling efficient swapping between main memory and secondary storage.
- Dynamic Memory Allocation: Efficient management of heap memory through functions like malloc() and free().

- Memory Mapping: Allows processes to map files or devices into their address space for efficient access to large data sets.

5. Distributed Systems and Networks

To enable "Draw It or Lose It" to communicate between various platforms:

- Implement a client-server architecture where the game server hosts core logic and manages game state.
- Use RESTful APIs over HTTPS for client-server communication, ensuring platform independence.
- Implement WebSocket connections for real-time updates during gameplay.
- Use a load balancer to distribute traffic across multiple server instances for improved performance and fault tolerance.
- Implement a distributed caching system (e.g., Redis) to reduce database load and improve response times.
- Use a message queue system (e.g., RabbitMQ) for asynchronous communication between different components of the system.

To handle connectivity issues and outages:

- Implement retry mechanisms with exponential backoff for failed requests.
- Use circuit breakers to prevent cascading failures in the distributed system.
- Implement a robust logging and monitoring system to quickly identify and resolve issues.
- Use database replication and regular backups to ensure data persistence and quick recovery in case of failures.

6. Security

To protect user information and ensure secure communication between platforms:

- Use HTTPS (SSL/TLS) for all client-server communications to encrypt data in transit.
- Implement strong user authentication using industry-standard protocols (e.g., OAuth 2.0, OpenID Connect).
- Use bcrypt or Argon2 for securely hashing and storing user passwords.
- Implement role-based access control (RBAC) to manage user permissions effectively.
- Use prepared statements and parameterized queries to prevent SQL injection attacks.
- Implement input validation and sanitization to prevent cross-site scripting (XSS) attacks.
- Use security headers (e.g., Content Security Policy, X-XSS-Protection) to enhance browser security.
- Regularly update all software components, including the OS, web server, and application dependencies.
- Implement a Web Application Firewall (WAF) to protect against common web exploits.
- Use intrusion detection and prevention systems (IDS/IPS) to monitor and block suspicious network activity.
- Conduct regular security audits and penetration testing to identify and address vulnerabilities.

By implementing these recommendations, The Gaming Room can ensure a secure, scalable, and efficient deployment of "Draw It or Lose It" across multiple platforms while meeting the specific requirements of the game and its users.