Tariq Mack

3/22/2025

Software Design Document Project 3 : Draw It or Lose It Web-Based Game

Executive Summary

The Gaming Room has requested a web-based version of their successful Android game, "Draw It or Lose It." As a Technology Consultant at Creative Technology Solutions (CTS), I am tasked with developing a scalable and maintainable software design that ensures efficient performance and cross-platform functionality. The core challenge is to transition the existing mobile-only game into a distributed, web-based environment while ensuring that game integrity, performance, and user experience are maintained. My proposed solution utilizes software design patterns and object-oriented programming principles to streamline development and ensure long-term scalability.

To address the requirement of having only one instance of the game active in memory at a time, I will implement the Singleton design pattern. This ensures controlled access to a single instance of the game service throughout the application lifecycle. Additionally, I will use the Iterator pattern to efficiently manage collections of teams and players and ensure the uniqueness of game and team names during creation. These design decisions will not only fulfill client requirements but also reduce system resource usage, improve maintainability, and support future enhancements.

Design Constraints

The game will be developed in a web-based distributed environment, introducing several design constraints:

Scalability: The system must support multiple concurrent users and teams while ensuring fast response times and minimal latency. The application should be able to scale horizontally as demand increases.

Single Instance Requirement: Only one instance of the game service can exist in memory at a time. This is critical for maintaining consistent game state and preventing duplication of game instances, which could result in synchronization issues.

Unique Identifiers: Each game, team, and player must have a unique identifier to prevent conflicts and ensure smooth gameplay. Name collisions must be avoided, which requires checks against existing names when creating new entities.

Cross-Platform Compatibility: The game must function consistently across various web browsers and operating systems. This limits certain platform-specific optimizations and requires adherence to web standards.

Security: Since the game will be web based, secure handling of user data and prevention of unauthorized access is essential. Communication between client and server must be encrypted.

Implications: These constraints guide the design choices in terms of technology stack, patterns used, and code architecture. For instance, Singleton ensures single-instance enforcement, while the Iterator supports collection management without exposing internal data structures. These patterns collectively improve performance, enforce rules, and allow for easier debugging and testing.

Domain Model

The UML diagram provided outlines a class hierarchy that leverages object-oriented principles such as inheritance, encapsulation, and abstraction. The base class Entity defines common attributes such as id and name, which are inherited by the Game, Team, and Player classes. This inheritance promotes code reuse and simplifies maintenance.

Game Class: Inherits from Entity and contains a list of Team instances.

Team Class: Inherits from Entity and contains a list of Player instances.

Player Class: Inherits from Entity and represents an individual player in a team.

GameService Class: Acts as a controller and manages the lifecycle of the game, teams, and players. It ensures the uniqueness of names and IDs using internal collections and provides methods to add or retrieve entities.

Design Patterns in Use:

Singleton Pattern: Applied in GameService to ensure that only one instance of the service exists in memory at a time. This is critical for maintaining a consistent state across the game.

Iterator Pattern: Used in methods like addGame() and getGame() to traverse collections efficiently and enforce name uniqueness. This ensures that no duplicate team or game names exist, fulfilling the client's requirements.

By using these patterns, the application design ensures efficient resource usage, scalability, and maintainability. The design allows for future enhancements, such as expanding to mobile web platforms or integrating with external databases, without requiring major refactoring.

# Evaluation Section

Software Design Document – Evaluation Section

Tariq Mack

Client: The Gaming Room  
Consultant: Creative Technology Solutions (CTS)  
Project: Expansion of "Draw It or Lose It" to a Multi-Platform Web-Based Application

Evaluation Table

Summary Recommendations

Based on the evaluation above, here are the recommendations for The Gaming Room's project expansion:

Server-Side Hosting:

Primary Recommendation: Linux is the best platform for hosting the web-based backend due to its stability, scalability, and zero licensing cost. It’s also cloud-friendly and integrates well with modern dev tools and infrastructure.

Windows Server can be considered if there’s a specific requirement for Microsoft technologies, but the cost and maintenance overhead is higher.

Client-Side Support:

A fully responsive web-based interface is the ideal approach. It ensures compatibility across all platforms—desktops (Linux, macOS, Windows) and mobile (Android and iOS).

A Progressive Web App (PWA) architecture is strongly recommended to provide app-like features such as offline mode and push notifications without needing separate native mobile development.

Development Tools and Team Structure:

Stick with open-source tools and languages (JavaScript/Node.js, Java, HTML/CSS, React) to keep costs low and development unified.

Only one cross-functional development team is necessary, but it should include:

Frontend specialists (UI/UX and responsive design experience)

Backend developers familiar with REST APIs and cloud deployment

QA testers with access to a wide range of devices and browsers

Platform-Specific Testing:

Mac users should test Safari regularly, as it can behave differently from Chrome and Firefox.

For mobile platforms, real device testing is preferred over emulators for performance and UI checks.

Budget-Friendly Practices:

Use free IDEs like VS Code and Android Studio unless team members already use licensed tools.

Minimize reliance on paid services unless it is essential to the app’s delivery.

# Recommendations Section

# Recommendations Section

Recommendations Section – Software Design Document

As the Technology Consultant for Creative Technology Solutions (CTS), I’ve spent the past several weeks evaluating the architectural needs of The Gaming Room’s Draw It or Lose It application. Based on our client’s goal to expand the game across multiple platforms, I’ve pulled together my recommendation for the most suitable operating platform and supporting technologies. Everything here is geared toward scalability, performance, reliability, and security across different environments.

Operating Platform Recommendation:

To support the expansion of Draw It or Lose It across various computing environments, I recommend adopting a Linuxbased server platform, specifically Ubuntu Server LTS. It offers longterm support, wide compatibility with crossplatform tools, and a strong developer community. Ubuntu Server can run on both local servers and virtual machines in cloudbased environments such as AWS, Azure, or Google Cloud. It’s opensource, costeffective, and incredibly stable, which is exactly what we need for a fastgrowing multiplayer game environment.

Ubuntu’s LTS versions receive security updates and support for up to five years, reducing maintenance overhead and ensuring reliability. Its commandline interface and compatibility with automation tools like Ansible and Chef also make it easier for our development and DevOps teams to script deployments and manage environments.

Operating System Architectures:

Ubuntu Server is built on the monolithic kernel architecture, which integrates process management, memory handling, device drivers, and file system management into a single large block of code running in a single address space. The benefit of this architecture is speed and efficiencyinterprocess communication happens quickly, and hardwarelevel operations are streamlined. Ubuntu supports 64bit architectures and works efficiently with ARM and x86based systems, making it a great fit for both highend servers and lightweight cloud environments.

Its modular nature allows us to remove unnecessary components, reducing the attack surface and improving both performance and security. The architecture is also wellsuited for containerized environments and supports features like cgroups and namespaces, making it ideal for microservices and cloudnative applications.

Storage Management:

For this project, I recommend using a combination of Logical Volume Management (LVM) and cloudbased object storage solutions like AWS S3 for unstructured game assets and playergenerated content. LVM will handle local disk partitioning and scalability on the Ubuntu server side, allowing for flexible disk resizing without system downtime. AWS S3 will manage largescale storage of drawings, player data, and backups.

This hybrid model lets us store systemcritical files locally for speed and reliability, while offloading nonessential or bulk content to a cloud environment for scalability and costeffectiveness. We can also implement RAID 10 for physical redundancy, and take advantage of AWS S3's lifecycle policies and versioning for automated backup and retention.

Memory Management:

Ubuntu uses virtual memory management through paging and demand paging techniques, which allows processes to run even when the physical memory is fully utilized. It also features a swap area that extends RAM with disk space when necessary. This is especially important for multiplayer gaming environments that experience memory spikes during peak traffic times.

The Linux kernel’s memory management system uses the Completely Fair Scheduler (CFS) to distribute CPU time fairly among running tasks, ensuring smooth gameplay and stable backend processing. Page caching and buffer optimization also enhance read/write operations for performance. Additionally, tools like vmstat, htop, and free will allow us to monitor memory usage in real time and optimize system performance with finetuned configuration.

Distributed Systems and Networks:

To support interplatform communication, we will deploy Draw It or Lose It in a distributed environment using a RESTful API architecture. Each client devicewhether Android, iOS, or webwill interact with the centralized server through stateless HTTP requests. To enhance communication between distributed components, we can incorporate message brokers like RabbitMQ or Kafka for eventbased communication.

These tools will help us manage realtime updates between users, synchronize scores, and handle game state changes efficiently. TCP/IP will be used for reliable transport, with HTTPS ensuring secure transmission. Load balancers like NGINX or HAProxy will distribute traffic evenly across server instances to ensure consistent performance.

We’ll use Docker containers to isolate services and deploy them on Kubernetes clusters for automated scaling and fault tolerance. Kubernetes will detect outages and redeploy failed containers automatically. This design will improve uptime and maintain service even during network interruptions or hardware failures.

Security:

Security is top priorityespecially in a multiplayer game environment that stores user data and manages accounts. Here’s how we’ll secure Draw It or Lose It across platforms:

1. Authentication: OAuth 2.0 and JWT (JSON Web Tokens) will be used to manage user sessions securely across different platforms. This ensures tokenbased authentication without exposing sensitive credentials.

2. Encryption: All data transmitted between clients and the server will be encrypted using HTTPS with TLS 1.3. Sensitive data at rest (e.g., usernames, email addresses, scores) will be encrypted using AES256.

3. Access Control: RoleBased Access Control (RBAC) will be used to restrict access to internal tools and admin functions. This ensures that only authorized personnel can access or modify key application components.

4. Firewall and IDS: We’ll implement iptables and UFW on Ubuntu for basic firewall rules, and Snort or OSSEC for intrusion detection. This protects against unauthorized access and monitors the system for anomalies.

5. Data Backup and Recovery: Automated backup routines will be set up using tools like rsync and cloudbased snapshots. These backups will be encrypted and stored across regions to support disaster recovery plans.

6. Monitoring and Alerts: With tools like Prometheus, Grafana, and Alertmanager, we’ll track key system metrics, set up realtime alerts, and respond proactively to potential threats like DDoS attacks or suspicious traffic.

By choosing a Linuxbased architecture supported by powerful memory and storage management tools, combined with scalable distributed system design and robust security protocols, The Gaming Room can confidently launch Draw It or Lose It across multiple platforms. This strategy ensures high availability, smooth user experience, and protection of sensitive dataall critical for supporting longterm growth and user trust.