**The Gaming Room**

**Software Design Template**

Version 1.2

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.2 | 10/17/2025 | Angel Silva | Revising Recommendation section to include more information specific to the client. |

**Executive Summary**

The gaming room company wants to design a new web-based game to serve across multiple platforms that is based on their current game Draw It or Lose It. There are four 1-minute rounds with images as clues, slowly being rendered until the 30-second mark. If the team at turn does not guess the puzzle, theopposing team gets one guess in 15 seconds to solve the puzzle.

**Requirements**

-Must allow for 1 or more teams.

-Each team will have multiple players.

-Game and team names must be unique for users to check if a name is available.

-Only 1 instance of the game can exist at one time.

-Must be web-based.

-Four 1-minute rounds.

-Pictures used as clues, gradually filling until the 30 second mark.

-Pictures represent a title, phrase, saying, etc.

**Design Constraints**

-Must allow for 1 or more teams.

-Each team will have multiple players.

-Game and team names must be unique for users to check if a name is available.

-Only 1 instance of the game can exist at one time.

-Must be web-based.

-Four 1-minute rounds.

-Pictures used as clues, gradually filling until the 30 second mark.

-Pictures represent a title, phrase, saying, etc.

**Recommendations**

The following section provides detailed recommendations for implementing The Gaming Room’s game Draw It or Lose It across multiple operating systems.

Operating Platform (Server)  
Recommend: Linux (Ubuntu Server LTS) running containers (Docker) behind an Nginx reverse proxy on a cloud provider (e.g., AWS EC2 or ECS/Fargate). Use managed PostgreSQL (e.g., RDS) and an in‑memory data store (Redis) for low‑latency state and pub/sub.  
  
**Why this fits The Gaming Room:**  
- Cost & licensing: No per‑core OS licensing; broad community and vendor support.  
- Performance: Production‑proven networking stack, efficient epoll/kqueue equivalents, strong WebSocket/SSE support for real‑time play.  
- Portability: Container images run identically on developer laptops, CI, and production nodes.  
- Security & DevOps: Mature ecosystem (AppArmor/SELinux, iptables/nftables, fail2ban, certbot, systemd, Nginx) and first‑class CI/CD tooling.  
  
**Operating System Architectures**  
Linux (Ubuntu LTS) architecture highlights:  
- Modular monolithic kernel: Core services (process scheduler, virtual memory manager, VFS, networking) in kernel space with loadable modules.  
- User space isolation: Processes communicate via IPC (pipes, sockets, shared memory). Namespaces (PID, net, mount, user) and cgroups partition resources—critical for containers.  
- VFS & pluggable filesystems: Unified POSIX interface across ext4/XFS and network filesystems (NFS/SMB).  
- Networking stack: Mature TCP/UDP/IP with epoll‑based async I/O; kernel TLS offload; iptables/nftables for filtering and NAT.  
- Init & services: systemd manages services, logging, and health checks.  
  
**Implication for Draw It or Lose It:**  
Game API and real‑time gateway (WebSocket) run in separate containers, scaled horizontally via load balancing. Linux kernel features isolate each service, and Nginx terminates TLS and routes traffic to healthy upstreams.  
  
**Storage Management**

Recommended storage layout:  
- Relational DB: PostgreSQL for durable state—users, game sessions, scores, leaderboards, purchases.  
- In‑memory cache/broker: Redis for ephemeral state, matchmaking queues, and pub/sub events.  
- Filesystem/object storage: ext4/XFS for container volumes; Object storage (e.g., S3) for static assets and backups.  
  
**Memory Management**  
Linux techniques used:  
- Virtual memory & paging: Each process has contiguous address space; kernel pages memory on demand.  
- Page cache & read‑ahead: Improves I/O for frequently accessed data.  
- Buddy allocator + slab/slub: Efficient allocation for kernel objects and user‑space requests.  
- NUMA awareness: Pin critical services for predictable performance.  
  
**App‑level guidance:**  
- Stateless API layer; session state in Redis or JWTs.  
- Pool DB connections; use async I/O to avoid memory bloat.  
- Monitor heap usage and autoscale based on latency and memory pressure.  
  
**Distributed Systems & Networks**  
Target topology:  
- Clients: Web (desktop/mobile) and native mobile apps.  
- Edge: CDN for static assets; WAF and TLS termination.  
- Gateway: Nginx routes to microservices.  
- Services: Game API, real‑time gateway, matchmaking, workers.  
- Data: PostgreSQL + Redis.  
  
**Cross‑platform communication:**  
- JSON over HTTPS; WebSockets for real‑time events.  
- Versioned APIs; retries with backoff.  
  
**Resilience:**  
- Circuit breakers, health checks, and graceful degradation.  
- Centralized logs, tracing, and metrics.  
  
**Security**  
User protection & platform security:  
- Identity & auth: Argon2id password hashing; JWT/OAuth2 tokens with rotation.  
- Transport: TLS 1.3, HSTS, and modern ciphers.  
- Data at rest: AES‑256 encryption for databases and object storage.  
- Application: Input validation, output encoding, CSRF protection, rate limiting, and CORS policy.  
- Secrets management: Vault or cloud KMS for API keys and credentials.  
- Least privilege: IAM roles and restricted network access.  
- Logging: Centralized audit trails; privacy compliance.  
  
**Final Recommendation Summary**  
- Platform: Linux (Ubuntu Server LTS) + containers, Nginx, PostgreSQL, Redis.  
- Why: Cost‑effective, portable, secure, and highly scalable.  
- Benefits: Strong OS architecture, memory management, storage, distributed communication, and end‑to‑end security for The Gaming Room’s Draw It or Lose It.