

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/18/25 | Michael Conrad |  |

**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](#_sbfa50wo7nsh)

Creative Technology Solutions (CTS) has engaged us to extend The Gaming Room’s existing Android-only title, Draw It or Lose It, into a web-based, multi-platform experience. The game loosely inspired by the TV show Win, Lose or Draw supports teams competing to guess a phrase, title, or thing from a rendered stock image. Each game consists of four one‑minute rounds, with drawings rendered at a steady rate and completed at the 30‑second mark; if the active team does not solve the puzzle in time, opposing teams may steal within a 15‑second window.

The proposed solution is a browser-accessible application backed by a stateless web service. It enforces unique game and team names, supports multiple players per team, and guarantees that only one instance of any game exists in memory at a time. The server exposes a REST/JSON API, and persists authoritative state in a relational database. We recommend Linux for server hosting due to cost, stability, tooling, and deployment maturity. Client delivery occurs entirely via standards-based web technologies (HTML5, CSS, JavaScript/TypeScript), making the experience available across macOS, Windows, Linux, iOS, and Android.

## Requirements

*• A game can have one or more teams.*

*• Each team supports multiple players.*

*• Game and team names must be unique to allow users to verify availability when creating or joining.*

*• Only one instance of any game can exist in memory at a time (authoritative singleton per game).*

*• Games run four one‑minute rounds; the drawing auto‑completes at 30 seconds; steal opportunities last 15 seconds.*

*Non-functional requirements include: cross‑platform web delivery, scalability for concurrent sessions, real‑time UI cues, secure transport (HTTPS), protection of user data, observability (logs/metrics), and automated CI/CD.*

## [Design Constraints](#_2et92p0)

Web‑based distributed environment: The solution must function reliably over the public internet with variable latency and intermittent connectivity. Therefore, server endpoints must be idempotent where practical, state changes must be atomic, and the client must gracefully handle retries and reconnection.

Single instance per game: Enforce an in‑memory game coordinator (per‑game singleton) guarded by a distributed lock to avoid duplicate instances across horizontally scaled nodes. Use unique, immutable identifiers for entities and validate unique names in the database with indexed constraints.

Name uniqueness: Backed by UNIQUE indexes on (game\_name) and (game\_id, team\_name). Provide instantaneous user feedback via an availability check endpoint to prevent race conditions during creation.

Real‑time cues: HTTP/REST for commands and queries; optional WebSockets for push updates (timers, drawing progress, steals). Fallback to long‑polling when WebSockets are unavailable.

Security & privacy: All traffic over TLS; session tokens via short‑lived JWT with refresh; server‑side authorization rules per game/team; input validation and rate limiting to block abuse.

## [System Architecture View](#_ilbxbyevv6b6)

The system follows a three‑tier web architecture: (1) Web client (browser), (2) Application layer (stateless API service and real‑time gateway), and (3) Data layer (relational database and cache). Horizontal scaling is achieved using containerized services orchestrated by Docker/Kubernetes behind a load balancer. A CDN serves static assets; an object store holds the stock image library.

## [Domain Model](#_8h2ehzxfam4o)

The UML for Draw It or Lose It centers on an abstract base class Entity that supplies common attributes (id, name, timestamps). Concrete types—Game, Team, and Player—extend Entity via inheritance. A Game aggregates one or more Teams (1..\*), and each Team aggregates one or more Players (1..\*). Game owns Rounds (four Round instances per Game) and Tracks the active Puzzle. Composition is used where lifecycles are bound (Rounds belong to a specific Game).

OO principles demonstrated: Inheritance (Entity → Game/Team/Player) reduces duplication of identifiers and naming logic; Encapsulation keeps invariants (e.g., unique names, single active round) behind methods; Composition and Aggregation model whole‑part relationships (Game→Teams, Team→Players); the Singleton pattern scopes the per‑game coordinator so that exactly one in‑memory instance governs timing and state transitions. Immutability of IDs prevents accidental aliasing; factories validate uniqueness before creation.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

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** | macOS can host web applications using Homebrew‑installed stacks (Nginx, Node.js/Java, PostgreSQL). Strengths include a polished developer experience and strong UNIX tooling. Weaknesses are licensing cost, limited headless/server SKUs, and fewer options for economical auto‑scaling in data centers. For production hosting, macOS is uncommon and cost‑inefficient. | Linux is the industry standard for hosting: excellent performance, security‑hardened options, first‑class container support, and abundant tooling. Commodity cloud pricing and autoscaling are mature. Weaknesses are infrequent driver friction on bare metal and variance across distros, but for containerized workloads these are minimal. Linux is the preferred server OS. | Windows Server provides strong enterprise integration (Active Directory, IIS) and good .NET hosting. It carries licensing costs and slightly higher baseline resource usage compared to minimal Linux images. Cross‑platform toolchains work, but container images are larger. Suitable if .NET/IIS are strategic; otherwise less cost‑effective than Linux for this workload. | Mobile devices should not host the server; instead they consume the web client. Hosting on mobile is impractical due to power, NAT, and availability constraints. Mobile considerations focus on efficient client delivery and low‑latency networking rather than hosting. |
| **Client Side** | Supporting macOS as a client is straightforward via standards‑based browsers (Safari, Chrome, Firefox). Development costs are low because a single web client serves all desktop OSes. Test effort must cover Safari quirks and accessibility compliance. | Linux desktop users access the same web client through Chromium/Firefox. Costs are limited to cross‑browser QA and packaging of any optional native helpers (not required here). | Windows support is delivered via Chromium/Edge/Firefox. Extra QA covers high‑DPI rendering and varied input devices. No additional implementation cost beyond cross‑browser testing. | iOS and Android are supported with a responsive web app. Additional effort is reserved for touch input, small‑screen layout, battery‑friendly timers, and mobile network variability. If desired, a thin native wrapper (Capacitor/Cordova) can enable push notifications. |
| **Development Tools** | Languages/Frameworks: TypeScript/Node.js (API), React (client), Java/Kotlin or C# are alternatives for the API. Tools: VS Code or JetBrains IDEA/WebStorm, Docker Desktop, Postman, Git, and Homebrew for dependencies. | Same language choices as macOS. CI/CD and runtime parity are excellent: Docker/Podman, systemd services, Nginx, PostgreSQL/MySQL, Redis, Grafana/Prometheus for observability. | Strong .NET option with Visual Studio and IIS/Kestrel for hosting; also supports Node.js/Java toolchains. WSL2 enables Linux‑like environments; Docker Desktop supports Windows and Linux containers. | For the browser client, standard web tools suffice. Optional native wrappers use Android Studio/Xcode with Capacitor. Automated testing via Playwright/Appium for mobile browsers. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**:

Operating Platform: Host server workloads on Linux in a cloud environment (containerized, autoscaled) and deliver the client as a web app to desktop and mobile browsers. This maximizes reach and minimizes cost while meeting concurrency and performance needs.

1. **Operating Systems Architectures:**

Operating Systems Architectures: Use a stateless microservice or well‑modularized monolith architecture running in containers behind a layer‑7 load balancer. One service exposes REST/JSON; an adjunct gateway provides WebSockets for real‑time events. Services are 12‑factor, readiness/liveness probed, and horizontally scalable.

1. **Storage Management**:

Storage Management: Use PostgreSQL for authoritative game state with normalized tables and UNIQUE constraints for names. Employ Redis for ephemeral data (timers, lobbies, sessions) and as a distributed lock manager to maintain a single in‑memory game coordinator per game.

1. **Memory Management**:

Memory Management: The server maintains lightweight per‑game coordinators with bounded caches; immutable IDs prevent copying; pooled object allocations and connection pools limit GC pressure. Redis or database persistence ensures recovery after restarts.

1. **Distributed Systems and Networks**:

Distributed Systems and Networks: Clients connect over HTTPS; commands go to REST endpoints; real‑time updates stream via WebSockets. A message broker (optional) fans out events to all team members. Implement exponential backoff, heartbeats, and replayable commands to tolerate outages.

1. **Security**:

Security: Enforce HTTPS, HSTS, and secure cookies; authenticate users with OAuth/OpenID and issue short‑lived JWT access tokens. Authorize access by game/team membership. Validate and sanitize all inputs; implement rate limiting and CSRF protections; encrypt data at rest and use least‑privilege IAM for services and storage.