Casino Blackjack (PWA) - Cloudflare-Native Architecture with Voice Controls

Full Product & Technical Design Document

v1.1 - August 08, 2025

Abstract

Complete end-to-end design for a globally-performant, low-cost Blackjack PWA with voice-first controls.

Includes product requirements, UX, architecture, data model, protocols, security, performance, observability, milestones, rollout, and success criteria.

1. Product Overview & Vision

Build a fair, accessible Blackjack experience that runs on desktop and mobile as an installable PWA.

Principles: (1) Edge-native and elastic, (2) Provably fair and transparent, (3) Accessible by default, (4) Simple, fast, and affordable.

2. Requirements

2.1 Functional Requirements

- Browse lobby; join/seats 1-7; spectators listen-only.
- Bet/deal/play actions: hit, stand, double, split, surrender, insurance (configurable).
- Authoritative server state and validation; deterministic replays.
- Provably fair shuffling with seed commit, round nonce, and later reveal; client verifier.
- Voice-first UX with push-to-talk (PTT) and TTS; full text parity.
- Installable PWA with offline shell and asset caching.
- Profiles and cosmetics; guest mode with limits.

2.2 Non-Functional Requirements

- Latency: WS RTT < 100ms typical; STT final <= 1.2s; TTS start < 400ms.

- Scale: 100s of concurrent tables, 1k+ players.
- Availability: >= 99.9% monthly; degrade to text-only if voice fails.
- Cost: serverless-first; compress payloads; keep hot state in-memory.
- Accessibility: WCAG 2.1 AA; captions; keyboard navigation; large touch targets.
- Internationalization: en-US first; extensible locale packs.
- Privacy: voice transcripts not retained by default; opt-in diagnostics.

2.3 Constraints & Assumptions

- No real-money wagering/KYC in v1.
- Mobile Safari STT limits; PTT-first with cloud STT fallback.
- KV is eventually consistent; use Durable Objects for authority.
- Edge CPU time is limited; design for short-lived invocations.

2.4 Out of Scope (v1)

- Voice chat between players.
- Full casino suite (future).
- Wake-word hands-free (deferred, explicit opt-in).

3. UI / UX

Personas: casual mobile, desktop player, host/streamer.

Navigation: Landing -> Lobby -> Table -> Profile/Settings. PWA install prompts shown contextually.

Design: Tailwind/shadcn components; Framer Motion for smooth transitions; color-blind friendly palette; high-contrast mode.

Voice UX: PTT button (spacebar/hold). Deterministic grammar; confirmations only on low confidence. Only final transcripts trigger actions. TTS announces state changes.

Accessibility: captions for all TTS; strong focus states; adjustable TTS voice/rate/pitch.

- Commands: join/leave/sit; bet/increase/clear; hit/stand/double/split/surrender/insurance; meta: repeat/mute/help/show rules.
- Errors: UI toasts + spoken prompts; never act on partial STT.
- Wireframe: table canvas, chip tray/bet inputs, action bar, PTT mic button, captions rail.

4. Architecture Overview

Cloudflare-native: Workers + Hono (REST/auth, WS upgrade); Durable Objects (TableDO per table, LobbyDO for presence); D1 for canonical persistence; KV for lobby snapshots; R2 for assets; Analytics Engine + Sentry for telemetry.

```
Client (React PWA, TS)
  <-> WebSocket (per table)
Hono Worker (REST/Auth, WS upgrade)
  <-> Durable Objects:
      - TableDO (authoritative table state, timers, replay buffer)
      - LobbyDO (presence & discovery; publishes snapshots to KV)
  -> D1 (users, tables, rounds, bets, hands)
  -> R2 (assets: cards, audio)
  -> KV (lobby snapshots, ephemeral)
  -> Analytics Engine / Sentry (metrics/errors)
```

5. Game Rules & Fairness

Rules: configurable decks/shoe, dealer S17/H17, split rules, double after split, surrender, insurance. A ruleset is bound to a table.

Provably Fair: per-shoe serverSeed commit (publish serverSeedHash before round 1). Each round uses nonce = roundIndex and latest clientSeed. Shuffle uses HMAC-SHA256-derived PRNG with rejection sampling for unbiased Fisher-Yates.

Anti-selective-reveal: append-only hash chain roundHash = SHA256(prevRoundHash || canonicalRoundSummary).

```
// Unbiased draw r in [0..i] using 32-bit words
const TWO32 = 0x100000000;
function drawInRange(nextU32, i){
  const m = i + 1;
  const bound = Math.floor(TWO32 / m) * m;
  let x; do { x = nextU32(); } while (x >= bound);
  return x % m;
}
```

6. Data Model (D1)

Money in integer chips; UTC timestamps (ms). Foreign keys on; unique constraints guard integrity.

```
PRAGMA foreign_keys=ON;

users(id TEXT PRIMARY KEY, handle TEXT UNIQUE NOT NULL, created_at INTEGER NOT NULL);
profiles(user_id TEXT PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE, avatar_url TEXT,
settings JSON);

tables(id TEXT PRIMARY KEY, name TEXT NOT NULL, status TEXT NOT NULL, created_at INTEGER NOT
NULL,
       shoe_id TEXT, server_seed_hash TEXT NOT NULL, current_nonce INTEGER NOT NULL DEFAULT 0);

seats(id TEXT PRIMARY KEY, table_id TEXT NOT NULL REFERENCES tables(id) ON DELETE CASCADE,
      user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE, position INTEGER NOT NULL,
```

```
      UNIQUE(table_id, position), UNIQUE(table_id, user_id));

rounds(id TEXT PRIMARY KEY, table_id TEXT NOT NULL REFERENCES tables(id) ON DELETE CASCADE,
       nonce INTEGER NOT NULL, shoe_id TEXT NOT NULL, server_seed TEXT, client_seed TEXT NOT
NULL,
       started_at INTEGER NOT NULL, revealed_at INTEGER, round_hash TEXT NOT NULL,
       UNIQUE(table_id, nonce));

bets(id TEXT PRIMARY KEY, round_id TEXT NOT NULL REFERENCES rounds(id) ON DELETE CASCADE,
     user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE, amount INTEGER NOT NULL
CHECK(amount > 0),
     placed_at INTEGER NOT NULL, UNIQUE(round_id, user_id));

hands(id TEXT PRIMARY KEY, round_id TEXT NOT NULL REFERENCES rounds(id) ON DELETE CASCADE,
      user_id TEXT REFERENCES users(id), cards TEXT NOT NULL, result TEXT,
      is_dealer INTEGER NOT NULL CHECK(is_dealer IN (0,1)));
```

7. Protocol & APIs

All WS/REST payloads JSON with zod validation; WS envelope carries seq for ordering. Clients send resume(lastSeenSeq) after reconnect to receive buffered deltas or a full snapshot.

REST: /api/tables, /api/profile, /api/seed-commit/:shoeId, /api/verify/:roundId, /api/lobby. WS: GET /ws/table/:id with short-lived JWT.

```
// Envelope
{ "seq": number, "type": string, "payload": any }

// Client -> Server
JoinTable { tableId, seat? }
LeaveTable {}
PlaceBet { amount }
Action { type:'hit'|'stand'|'double'|'split'|'surrender'|'insurance' }
Ready {}
SetClientSeed { clientSeed }
VoiceMeta { enabled:boolean, locale:string }
Ack { upTo:number }

// Server -> Client
TableStateFull { seats, minBet, maxBet, shoeCount, phase, timers, seq }
Delta { patch }
Phase { name, deadlines }
BetAccepted / BetRejected
Dealt { hands, upcard }
Turn { userId, deadline }
Outcome { payouts[], dealerHand }
SeedHash { serverSeedHash }
SeedReveal { serverSeed, clientSeed, nonce }
ReplayGap { from, to }
Error { code, message }
```

8. Security & Compliance

- OAuth via Lucia; WS with short-lived JWT (5-10 min) scoped to session, tableId, ipHash; same-origin required.
- REST CSRF tokens; cookies HttpOnly, Secure, SameSite=Lax.
- Cloudflare Turnstile for guest sign-in.
- Per-user and per-IP token buckets; backoff on reconnect; flood protection.
- Enforce phase + seat ownership server-side.
- CSP + HSTS; sanitize inputs; secrets in Cloudflare Secrets.
- Privacy: voice transcripts off by default; opt-in diagnostics; age-gate and regional disclaimers via flags.

9. Performance & Capacity

- Keep shoe/hot state in TableDO; persist summaries post-round.
- Deltas or small events (<2KB typical); enable compression or MessagePack.
- Timers via DO Alarms for resilience across restarts.
- Targets: WS RTT < 100ms; STT final <= 1.2s; TTS start < 400ms.
- Batch broadcasts; apply backpressure; cut misbehaving clients.
- Regionalize STT; aggressively cache static assets.

10. Observability

- Structured logs: {ts, requestId, tableId, roundId, userId, seq, event}.
- Metrics: active_tables, avg_rtt, stt_confidence p50/p95, round_duration, errors_by_code, reconnect_rate.
- Alerts: p95 RTT thresholds, DO restart spikes, D1 error rates, WS disconnect storms.
- Dashboards: gameplay funnel, fairness audits, cost per 1k player-mins.

11. Developer Experience & Tooling

- Monorepo with pnpm/turbo.
- Packages: game-core (rules/shuffle/RNG), protocol (zod), voice (grammar/parsers).
- Local dev: Miniflare for Worker + DO + WS; deterministic fixtures.
- CI: Actions -> typecheck -> unit -> integration -> e2e -> wrangler publish.
- Standards: strict TS, eslint/prettier, conventional commits.

12. Implementation Plan & Milestones

Each phase has exit criteria and demoable deliverables.

- Phase 0 (W1-2): Scaffold; Worker + DO echo; PWA shell; PTT stub. Exit: connect to test table and hear TTS.
- Phase 1 (W3-6): Rules; bets->deal->play->payout; per-shoe commit; audit hash chain; text UI. Exit: two clients play full rounds deterministically.
- Phase 2 (W7-9): Full voice parity; TTS announcements; reconnect resume (seq/ack). Exit: recover from WS drop without desync.
- Phase 3 (W10-12): LobbyDO, profiles, cosmetics, moderation; analytics; load testing. Exit: 200 players across tables with p95 RTT < 120ms.
- Phase 4 (W13+): Additional games reusing DO patterns.

13. Rollout Plan

- Environments: dev -> staging -> prod with separate bindings.
- Feature flags: LLM NLU fallback, wake-word.
- Canary: 5% for 24h, then ramp; rollback via wrangler versions.
- Runbooks: WS incident, DO hot-shard, D1 migration failure, STT vendor failover.
- Error budgets: auto-disable LLM fallback on threshold breaches.

14. Success Criteria & KPIs

- Activation: >= 60% of new users complete one round in first session.
- Reliability: >= 99.9% monthly; resume success >= 98%.
- Performance: p95 WS RTT <= 120ms; STT final <= 1.2s; TTS start < 400ms.
- Fairness: 100% rounds verifiable; RNG bias tests pass.
- Cost: <= target $/1k player-mins (finalize after load tests).
- Accessibility: WCAG audits pass; manual QA on screen readers.

15. Risks & Mitigations

- Mobile Safari STT inconsistency -> PTT-first, server-side streaming fallback, full text parity.
- WS flakiness -> seq/ack + replay buffer; full-state resync; backoff and resume.
- Lobby staleness -> LobbyDO as authority; KV for snapshots only.

- Fairness skepticism -> public docs + verifier; hash-chain audits; seed rotation.
- Abuse/spam -> Turnstile, quotas, per-IP/user rate limits, phase allowlists.
- Vendor lock-in -> abstract STT/TTS; keep core logic in portable TS packages.

16. Appendix

A. Voice grammar examples (en-US/en-GB).

B. Verifier snippet and round audit chain details.

C. wrangler.toml excerpt.

D. Test matrix (browsers/devices/network).

E. Sequence diagrams for join/play/resume.

```
wrangler.toml (excerpt)
name = "blackjack-worker"
main = "src/index.ts"
compatibility_date = "2025-08-01"

[durable_objects]
bindings = [ { name = "TABLES", class_name = "TableDO" } ]

[[d1_databases]]
binding = "DB"
database_name = "blackjack_db"

[[kv_namespaces]]
binding = "LOBBY"; id = "..."

[[r2_buckets]]
binding = "ASSETS"; bucket_name = "blackjack-assets"
```