# Mobile “Offline-Only” BBQ Competition Server — Product Specification (v0.1)

**Purpose:** Provide a self-contained, event-ready, offline web platform to run BBQ competitions end-to-end (setup → registration → turn-in → judging → scoring → reporting) **without relying on external Wi-Fi or Internet**. All participants operate **only** on the event network, and all data is stored locally with **near-zero tolerance for downtime or corruption**.

---

## 1) Non-Negotiable Requirements

### 1.1 Availability & Data Integrity

- **Database must not go down** during an event. A DB outage is an event failure.
  - **Database must not get corrupted** (power loss, storage failure, process crash must not cause irrecoverable data loss).
  - System must tolerate:
    - Single node failure (server/worker Pi failure)
    - Single SSD failure (in the database layer)
    - Power interruption long enough to complete safe shutdown (UPS-driven)

**Definition of success:** The event can complete and produce final rankings and team reports even if one compute node fails mid-event.

### 1.2 Offline-Only Network / “Walled Garden”

- All client devices must use the event network and reach only:
    - the load balancer
    - the application services
    - DNS/captive portal services (if enabled)
  - Internet access is blocked by default (optionally allowed for admins only).

### 1.3 Event Simplicity & Operator UX

- Host arrives, flips **one master power switch**.
  - Cluster self-boots into “Ready” or “Not Ready” state.
  - Physical indicators:
    - Per-node LED (Green = healthy, Red = unhealthy)
    - Master LED (Green = cluster ready, Red = cluster not ready)
  - Minimal steps to start an event:
    - “Create Event”

    - "Configure Categories / Rules / Weighting"
    - "Open Registration"
    - "Open Turn-in"
    - "Start Judging"
    - "Finalize / Publish Results"
    - "Generate Reports"

---

## 2) System Overview

**2.1 Roles**

- **Host/Admin (staff):** Configure competition, manage check-in, validate turn-ins, orchestrate judging flow, finalize results, print/export reports.
  - **Judge:** Uses personal device (phone/tablet) to score assigned samples. Must be guided; should not choose categories manually.
  - **Runner/Table Host (staff):** Physically distributes samples and coordinates "appearance" then "taste/texture" rounds per table.
  - **Contestant (optional access):** If enabled, can view their own reports after completion (or staff prints them).

**2.2 Core Workflows**

1. **Event Setup**
- Define categories (e.g., chicken, pulled pork, ribs, brisket)
    - Define grading criteria (appearance, taste, texture)
    - Define grading scale (7–10 with labels)
    - Define weighting (per criterion and per category)
    - Define number of judge tables and seats per table

2. **Registration**
- Team registers
    - System assigns unique Team ID
    - System produces scannable team code (Aztec/PDF417 recommended)
    - Team code is printed and given to team

3. **Turn-in**
- Each submission has a pre-printed identifier (submission code)
    - Staff scans submission code + scans team code to validate ownership
    - System records timestamp and marks submission "Received"

4. **Judging**
- Judges scan table QR to open table-specific URL
    - Judge enters seat number (1–6)

    - Server assigns the judge's next sample, consistent with the table's distribution plan
    - Flow:
        - **Appearance phase:** judges evaluate all assigned samples in numeric order (1..N at table)
        - **Taste/Texture phase:** judges evaluate in seat-specific "passing order"
    - Comments allowed for each criterion

5. **Scoring & Results**

- System calculates per-team category score
    - Produces ranking best → worst for each category
    - Calculates overall winner using host-defined category weights
    - Produces team report including aggregates + all comments

---

## 3) Hardware Architecture (High Availability, Offline, Event-Proof)

### 3.1 Guiding Principle

**Do not use Raspberry Pi as the Wi-Fi access point for 200 clients.** Use proper AP hardware and keep Pis as compute/storage.

### 3.2 Recommended Physical Stack (Minimum "No DB Downtime")

This is the baseline that matches the "DB cannot go down" requirement.

**A) Network / Edge**

- 1× **Router/Firewall appliance** (small x86 or high-quality travel router with VLAN + firewall rules)
  - 2× **Wi-Fi Access Points** (capacity planning for 200 devices; dual-band, tuned for high client count)
  - Optional: managed switch (PoE if APs need it)

**B) Compute Cluster**

- 2× **Load Balancer Nodes**
    - Can be Raspberry Pi 4/5
    - Runs HAProxy/Nginx + Keepalived (Virtual IP failover)

- 2–3× **Application Worker Nodes (RPi 5 recommended)**
    - Runs Node.js API and serves the SPA static site
    - Horizontal scale: add more workers as needed

**C) Database Layer (Hard Requirement)**

- 2× **Database Nodes** (dedicated)

        - Each with **USB3 SSD** (high endurance)
        - Runs PostgreSQL in an HA configuration with synchronous replication
    - 1× **Witness/Quorum Node** (lightweight)
        - Prevents split-brain
        - Can be a small Pi or run on one LB node if isolated appropriately

**Total:** 2 LB + 2 DB + 1 Witness + 2–3 App Workers = **7–8 devices**

(You can combine Witness onto an LB node if you must reduce count → **6–7 devices**, but separate is cleaner.)

### 3.3 Power & Corruption Protection

- **UPS is mandatory** for DB integrity:
        - UPS powers router/switch/AP + DB nodes + at least one LB and one App worker
        - UPS triggers clean shutdown on low battery
    - **Storage:**
        - DB nodes use SSD only; **no microSD for DB data**
        - Enable filesystem and DB settings for durability (see §7.3)
    - **Enclosure:**
        - Rugged case with airflow
        - Labeling for ports and node roles
        - Single master toggle switch controlling a power strip feeding all components

### 3.4 LED Health Indicator System

- Each node exposes a local health endpoint or exports status to the “Indicator Controller”.
    - **Indicator Controller** (can be one LB node or a dedicated small Pi):
        - Polls:
            - LB active/standby status
            - App workers health (HTTP 200 + dependency checks)
            - DB cluster leader/replica health + replication lag
            - Disk SMART + filesystem read/write checks
            - UPS status (battery/line power)
        - Drives:
            - Per-node LED: Green/Red
            - Master LED:
                - Green if minimum quorum healthy:
                    - (LB VIP is up)
                    - (≥1 App worker healthy)
                    - (DB leader healthy AND replication healthy OR failover-

ready)
        - Red otherwise

---

## 4) Network Design ("Only on Our Network")

### 4.1 SSID & Client Access

- 1 SSID for judges/participants (WPA2/WPA3 PSK)
  - Optional separate SSID for staff/admin devices

### 4.2 DNS & URL Experience

- Preferred: https://event.local/ or http://event.local/
  - Local DNS server on router or cluster resolves event.local to the LB virtual IP
  - Optional captive portal to auto-open the judging web app upon connection

### 4.3 Firewall Rules

- Block outbound Internet for participant SSID VLAN
  - Allow participant VLAN → LB VIP ports only (80/443)
  - Staff VLAN can reach admin endpoints
  - Optional: block device-to-device traffic on participant VLAN (client isolation)

---

## 5) Software Platform Recommendations

### 5.1 Front-End

- **React** (recommended) or Angular
  - Must be **mobile-first** and work on:
    - iOS Safari
    - Android Chrome
  - Implement as **PWA** for:
    - fast reload
    - resilience to brief Wi-Fi drops
    - optional "Add to Home Screen"
  - Keep UI simple and "guided":
    - Judges never choose category; server assigns next step.

### 5.2 Back-End API

- **Node.js** with a structured framework:
    - **NestJS** recommended (strong modularity, validation, DI)
  - API patterns:

    - REST for most operations
    - Optional WebSockets/SSE for live status dashboards (host view)

### 5.3 Database

- **PostgreSQL** (primary choice)
  - HA Layer:
    - Patroni (or similar) for leader election and automated failover
    - Synchronous replication between DB nodes
    - Quorum/witness to prevent split-brain

### 5.4 Cache / Queue (Optional but Strongly Recommended)

- Redis for:
    - rate-limiting
    - short-lived session tokens
    - buffering telemetry/metrics
  - If you can accept "acknowledged then persisted" semantics:
    - message queue for smoothing spikes
    - BUT final persistence must be in Postgres

---

## 6) Data Model (Conceptual)

### 6.1 Entities

- **Event**
    - id, name, date, location, status (setup/registration/turnin/judging/finalized)
  - **Category**
    - id, event_id, name (chicken, ribs, etc.), enabled
  - **Criteria**
    - appearance/taste/texture (baseline), configurable list
  - **Scale**
    - values 7–10 with labels (poor/fair/good/excellent)
  - **Weighting**
    - per-category criterion weights (e.g., taste 0.5, texture 0.3, appearance 0.2)
    - overall category weight for grand champion calc
  - **Team**
    - id, event_id, team_number, team_name, metadata
  - **TeamCode**
    - team_id, code_type (Aztec/PDF417), payload, printed_at
  - **Submission**
    - id, event_id, category_id, submission_number, submission_code, status

  - **TurnIn**
    - submission_id, scanned_team_id, verified (bool), timestamp, operator_id
  - **JudgeTable**
    - id, event_id, table_number, seats (fixed at 6 in this spec)
  - **JudgeSeat**
    - table_id, seat_number (1–6), active_session_token, judge_alias (optional)
  - **AssignmentPlan**
    - category_id, table_id, list of submission_ids assigned to that table
  - **Score**
    - submission_id, judge_seat_id, criterion_id, value (7–10), comment, timestamp
  - **AuditLog**
    - append-only: who did what, when, from which device

### 6.2 Integrity Constraints

- One score per (submission_id, judge_seat_id, criterion_id, phase)
  - Turn-in must record both scanned submission + scanned team, and verify team ownership
  - Submission cannot be judged until status = "AssignedToTables" and judging phase is active

---

## 7) Judging Logic & Sequencing

### 7.1 Table Assignment

Inputs:

- number_of_judges = tables × 6
  - total_submissions per category
  - Output:
  - Each table receives ⌈total_submissions / tables⌉ items (balanced as evenly as possible)
  - Randomization:
    - Randomize submissions to tables
    - Randomize initial order within each table list (but assign a stable numeric order for appearance phase)

### 7.2 Appearance Phase

- Judges evaluate appearance for **all submissions at their table**
  - Order: numerical (1..N) within that table's list
  - Each score:
    - value 7–10

    - optional comment
  - System enforces completion gating:
    - Taste/texture phase cannot start until appearance complete for the table (or per-judge; choose one policy and enforce consistently)

**7.3 Taste/Texture Phase**

- Judges evaluate taste and texture per submission, following the seat-specific "passing order"
  - The system must compute the next submission for each seat based on:
    - Table submission count N
    - Seat number s (1–6)
    - A defined deterministic permutation that matches physical passing behavior

**Requirement:** The seat ordering algorithm must reproduce the patterns described:

- Example for seat 1: 1, 7, 8, 9, 10, 11, 12, 13, 14, 15, 6, 5, 4, 3, 2
  - Example for seat 6: 6, 5, 4, 3, 2, 1, 15, 14, 13, 12, 11, 10, 9, 8, 7

Implementation approach:

- Represent table seats in a ring and define a "pass direction"
  - Model "batches" of 6 (each round of passing)
  - Generate a sequence per seat as a function of:
    - initial seat assignment
    - pass direction
    - serving entry point (seat 3 per your description)
  - Store the generated sequence per (table, category) so it is stable and auditable.

**7.4 Comments**

- Comments allowed for each criterion on each submission
  - Host reports must include all comments (with judge anonymization if desired)

---

## 8) Scoring & Reporting

**8.1 Per-Submission Scoring**

For each submission:

- Aggregate per criterion across 6 judges:
    - mean or trimmed mean (choose policy)
  - Weighted criterion score:
    - criterion_score × criterion_weight

**8.2 Per-Team Category Score**

- Team score for category = submission score (assuming one submission per category)
  - If multiple submissions per category are possible, define policy (best-of, average, etc.)

**8.3 Overall Score**

- Overall team score = sum(category_score × category_weight)
  - Output:
    - category rankings
    - overall rankings

**8.4 Team Report**

Per team:

- Per category:
    - appearance avg + comments
    - taste avg + comments
    - texture avg + comments
    - overall category score
  - Overall placement (if finalized)

Exports:

- PDF report per team
  - CSV export of all scores (for audit/recovery)

---

## 9) Security & Anti-Tamper

**9.1 Codes (Team & Submission)**

- Use **Aztec or PDF417** codes for printed identifiers
  - Payload should be:
    - opaque token (UUID + signature)
    - not human-readable IDs
  - Include a signature/HMAC so codes can't be forged offline

**9.2 Judge Access**

- Table QR encodes:
    - table_id
    - signed token with expiry
  - Judge enters seat number (1–6)
  - Server issues seat session token:
    - short-lived access token

    - renew silently
  - Prevent cross-seat scoring:
    - seat token binds to seat_id; API rejects mismatch

### 9.3 Admin Access

- Admin UI available only on staff VLAN (recommended)
  - Two-level permissions:
    - “Operator” (turn-in, table management)
    - “Administrator” (setup, finalize, exports)

---

## 10) Reliability Engineering (“DB Cannot Go Down”)

### 10.1 Database HA Strategy

- Two Postgres nodes with **synchronous replication**
    - Leader commits only when replica confirms write (prevents data loss on leader failure)
  - Automated failover with Patroni (or equivalent)
  - Quorum/witness to avoid split-brain
  - Health checks enforced at LB and app tiers:
    - If DB is degraded, app shows “pause” state rather than accepting writes blindly

### 10.2 Storage Strategy

- Each DB node uses:
    - SSD with high endurance
    - Proper filesystem (ext4 recommended) with safe mount options
  - Optional but strong:
    - Mirror SSDs (RAID1) per DB node if hardware supports it (often easier on x86 than Pi)

### 10.3 Power Strategy

- UPS required.
  - Auto graceful shutdown on low battery.
  - On boot:
    - DB cluster checks consistency
    - App stays in “Not Ready” until DB leader elected and replication healthy

### 10.4 Backup Strategy During Event

- Continuous:
    - write-ahead log archiving to the second DB node
  - Periodic (e.g., every 2 minutes):

    - logical snapshot export to a third storage location (USB SSD on controller)
  - End-of-round:
    - export CSV + PDF packs
  - Outcome:
    - Even in catastrophic failure, you have a near-real-time recovery point.

---

## 11) Implementation Plan (Phased Rollout)

**Phase 0 — Prototype (Single Node)**

- One machine with Postgres + Node + React
  - Implement:
    - event setup
    - registration + codes
    - turn-in verification
    - basic judging UI
    - scoring + reports

**Phase 1 — Production MVP (No-Downtime DB Baseline)**

- Introduce:
    - LB VIP failover
    - 2 app workers
    - 2 DB nodes synchronous replication
    - UPS + health indicator controller
  - Add:
    - audit logs
    - admin/staff roles
    - exports (CSV/PDF)

**Phase 2 — Event Operations Polish**

- Captive portal / easy “connect & judge”
  - Host dashboards:
    - turn-in progress per category
    - judging completion by table/seat
    - “stuck judge” detection
  - Better randomization + repeatability (seeded randomness)

**Phase 3 — Hardening**

- Fault injection tests:
    - kill app node
    - kill DB leader
    - unplug SSD

    - power-loss simulation on UPS
  - Load testing for:
    - 200 concurrent judges
    - 30–60 writes/sec bursts

---

## 12) Acceptance Criteria

**Availability / Integrity**

- If any single app worker dies, system continues without loss.
  - If DB leader dies, replica is promoted automatically and writes continue.
  - If power fails:
    - UPS keeps system alive long enough to shut down cleanly.
    - On restart, event data is intact and resumes.

**Performance**

- Supports:
    - 200 concurrent clients
    - typical 5–10 API requests per device per minute
    - burst handling during “save scores” moments

**Operator UX**

- “Power on → Ready” within a defined boot window (target: <5 minutes).
  - Master LED accurately signals readiness.
  - Judges can score with minimal steps:
    - scan → seat → score

---

## 13) Open Decisions (Project Manager Must Lock Early)

1. Final DB HA tooling: Patroni vs another approach.
  2. Trimmed mean vs average scoring policy; tie-break rules.
  3. Whether taste/texture scoring is per-submission sequential locking or allows out-of-order entry.
  4. Judge anonymity policy in team reports.
  5. Captive portal requirement vs simple DNS URL.

---

## 14) Deliverables Checklist (Engineering)

- Infrastructure-as-code for node provisioning
  - Docker images / compose files for:
    - LB + VIP
    - App worker
    - DB HA stack
    - Monitoring + LED controller
  - Full schema migrations + seed data

  - Admin UI + Judge UI + Operator UI
  - PDF generation service and templates
  - Load tests + chaos tests
  - Event day runbook:
    - setup checklist
    - troubleshooting LED states
    - emergency fallback steps

---

## 15) Event Day Runbook (High-Level)

1. Place enclosure, plug in, flip master switch.
2. Wait for master LED = Green.
3. On admin device: open event.local/admin.
4. Create/select event; configure categories/weights.
5. Start registration; print team codes.
6. Start turn-in; scan submissions and teams.
7. Lock turn-in; generate table assignment plan.
8. Start judging:

- table QR posters displayed
  - judges scan and enter seat

9. Monitor completion dashboard; resolve stuck seats.
10. Finalize scoring; publish results; generate reports.
11. Export full event archive (CSV + PDFs) to external storage.

---