CricAlgo — Product Requirements Document (PRD)

---

1. Overview / Goal

CricAlgo is an invite-only Telegram bot + admin web dashboard that enables a closed group of users to register, deposit USDT (BEP20), join contests tied to matches, win prizes, and withdraw winnings. The product is intended as a developer learning project and will be built as a production-capable system with manual finance controls handled by a single super-admin.

Primary goals: - Provide a smooth, secure Telegram UX for contest discovery and joining. - Provide an admin web dashboard for financial approval, contest management and reconciliation. - Maintain clear on-chain and off-chain accounting with per-user wallet balances split into Deposit / Winning / Bonus sections.

---

2. Core features (Must-haves vs Nice-to-haves)

Must-haves (launch)

- Invite-only registration (multi-use codes which admin can disable/expire).
- Username-based user accounts (Telegram ID + username), editable later.
- Admin web dashboard: user management, invite code management, deposit approval queue, withdrawal queue, match & contest CRUD, transaction logs, CSV export.
- Bot UX: Start -> Invite -> Create username -> Main menu (Profile, Matches, Balance).
- Deposit flow: single shared platform BEP20 address; user submits tx hash; admin reviews and manually marks deposit approved.
- Wallet model: per-user 3-bucket balances — Deposit, Winning, Bonus. Withdrawals only allowed from Winning.
- Contest creation: entry fee, max players, contest type, join cutoff, unique contest code, prize distribution rules, admin-set platform commission.
- Join flow: user joins via bot; on success user receives unique entry code.
- Winner declaration: admin selects winner(s) in dashboard; system computes payouts automatically and credits Winning balances.
- Cancellation: admin cancels contests -> automatic full refund to user Deposit wallet.
- Transaction history (30 most recent transactions visible in bot) and notifications for deposit approval, contest start, winners, withdrawal status.
- Security basics: admin authentication + 2FA, rate limiting for joins, audit logs.
- Backups & export: database backups and CSV exports for reconciliation.

Nice-to-haves (post-launch)

- Automated blockchain monitoring (watcher) to auto-detect deposits and propose approvals.
- Multisig / custody provider integration for withdrawals.
- KYC flows for withdrawals.
- Referral payouts distributed automatically in USDT or bonus.

- Web-based join links and public contest pages.
- Role-based admin users (finance, moderator) and approval workflows.
- Staging environment + CI/CD pipelines and automated tests.

---

3. Target Users & Roles

- **End user**: Registers by invite, deposits USDT, joins contests, views balances and history, receives notifications, requests withdrawals.
- **Super-admin**: Single admin account with full access: approve deposits, process withdrawals, create matches/contests, manage invite codes, run reports.

---

4. User Flows (examples & edge cases)

4.1 Registration (bot)

1. User opens bot -> `/start` -> bot prompts: "Enter invitation code".
2. User submits code. If valid & enabled -> ask for desired username.
3. Username validated for uniqueness & allowed characters; account created (store Telegram ID + username + createdAt).
4. Bot lands user in main menu.

Edge cases: - Invalid/disabled/expired code -> user shown friendly message and contact admin. - Duplicate username -> prompt to pick another.

4.2 Deposit (bot -> admin)

1. User navigates Balance -> Deposit -> Bot shows platform BEP20 address and deposit instructions (include exact memo/format if used).
2. User sends USDT to shared address off-chain and returns to bot -> Submit TX hash.
3. Bot records deposit request in `deposit_queue` with txHash, amount (user-entered), chain=BEP20, timestamp.
4. Admin reviews deposit queue and confirms on-chain receipt (manual). On approval, system credits user's Deposit wallet and notifies user.

Edge cases: - User submits wrong token or wrong chain txHash -> admin resolves manually. - User submits invalid txHash -> show validation error and allow re-submit. - Double-submitted tx -> system deduplicates by txHash.

4.3 Join contest

1. User opens Matches -> selects Match -> sees list of Contests.
2. Clicks a contest -> shows contest details with entry fee, max players, join cutoff, prize structure, available seats.
3. User clicks Join -> system checks balance ordering: use Deposit + Bonus first (in that order), then Winning if needed.

4. If funds sufficient -> create Entry record (user, contest, entryCode, timestamp) and debit amounts from user's sub-balances accordingly; notify user with entryCode.

Edge cases: - Concurrent joins hitting the last seats -> server-side optimistic locking (DB-level transaction) to prevent over-subscription. - Insufficient funds -> friendly error with top-up instructions.

4.4 Declare winners & payout

1. Admin navigates contest -> chooses winner(s) (single or multiple as defined).
2. Platform computes payout per contest rules, applies platform commission, and credits `Winning` balances for winners (automated calculation). System records transaction logs.
3. Notify winners & update contest status to Closed.

Edge cases: - Dispute: admin can reverse credits (manual amendment) with reason logged.

4.5 Withdrawal

1. User requests withdrawal from Winning balance: provides target BEP20 address and amount.
2. Admin sees request in withdrawal queue, validates off-chain, executes on-chain send from platform wallet, then marks request as Paid with optional txHash.
3. On marking Paid, system debits user's Winning balance and logs transaction.

Edge cases: - Admin marks Paid but tx fails -> admin must mark Failed and refund Winning balance. - Duplicate requests -> server-side idempotency.

---

5. Admin requirements (dashboard)

Minimum modules at launch: - **Auth & admin profile:** secure login + 2FA, last-login audit, password reset. - **Invite Codes:** create, multi-use, expiry toggle, disable/reactivate, usage stats. - **User management:** search, view balances (Deposit/Winning/Bonus), edit username, manual credit/debit, freeze account. - **Deposit queue:** list pending deposits with txHash, user claim, quick link to block explorer, approve/reject with notes. - **Withdrawal queue:** list requests, approve/mark-paid/failed with optional txHash, single-admin flow. - **Matches & Contests:** create/edit (no editing after users joined), set prize rules, entry fee, platform commission. - **Entry list:** view all entries per contest; export CSV. - **Reports & Exports:** transactions, user balances, contest P&L (CSV). - **Audit Logs:** record admin actions (who/when/what) for deposit approvals, payouts, balance changes.

UI expectations: responsive, role-ready for future RBAC expansion.

---

6. Data model (high-level)

Key entities (fields abbreviated): - `users`: id, telegram_id, username, created_at, status(frozen/active) - `wallets`: user_id, deposit_balance, winning_balance, bonus_balance - `transactions`: id, user_id, tx_type(deposit/withdraw/join/payout/refund/fee), amount, currency, related_id, timestamp, metadata - `invitation_codes`: code, max_uses/null, expires_at/null, enabled, created_by - `matches`: id, title, start_time, status - `contests`: id, match_id, code, entry_fee, max_players, prize_structure(json), commission_pct, join_cutoff, status - `entries`: id, contest_id, user_id, entry_code, amount_debited, created_at - `deposit_requests`: id, user_id, tx_hash, amount_claimed, chain, status - `withdraw_requests`: id, user_id, to_address, amount, status, admin_tx_hash - `audit_logs`: actor_admin_id, action_type, details, timestamp

DB recommendation: **PostgreSQL** (relational constraints, transactions, good concurrency control). Use UUID PKs for public references.

---

7. Integrations & APIs

- **Telegram Bot API** (main UX) via webhook or long-polling.
- **BEP20 blockchain explorer API** (optional future) for deposit verification (BscScan API) — optional in V1.
- **Crypto wallet/operator**: manual hot wallet controlled by admin. Provide interface to store/send transactions via simple admin UI for recording txHash.
- **Email / Notification**: optional SMTP for admin alerts; initial notifications via Telegram messages.
- **Storage**: object storage (S3-compatible) for backups and optional screenshots.

---

8. Security & key management

- Admin access: Password + mandatory 2FA (TOTP). Session expiry and IP-based session invalidation feature recommended.
- Store secrets in a secrets manager (env vars are acceptable for dev, but use Vault/Secrets Manager in prod).
- Custody: hot wallet controlled by you (private keys remain offline on your hardware). Document withdrawal procedure and require manual reconciliation.
- Rate limiting: bot endpoints / join actions limited per user and per IP.
- Entry concurrency: DB transactions + row-level locking when decrementing available seats.
- Logging & audit: immutable audit logs for financial actions with admin id, timestamp and reason.
- Backups: daily DB backups with retention policy (configurable), test restore quarterly.

---

9. Performance, scale, and availability

- Initial target: support ~50 DAU and ~50 concurrent users smoothly.
- SLA: 99.9% availability. Design with stateless app servers behind load balancer, single DB instance with read-replicas if needed.
- Use connection pooling (PgBouncer) and optimistic pagination for lists.
- Plan for spikes at match start — add queueing and graceful failure messaging.

---

10. Non-functional requirements

- Logging: structured logging (JSON) for server logs, stored centrally.
- Monitoring & Alerts: Sentry (errors), Prometheus/Grafana (metrics), alerts to Telegram or Email.
- Rate limiting & throttling: global and per-user limits.
- Localization: messages in English; design to allow future i18n.
- Data retention: keep records indefinitely, but provide admin tools to export and purge per policy.

---

11. Business rules & wallet behavior (important)

- Wallet buckets: Deposit, Winning, Bonus.
- **Deposit**: funds user contributed; used first for contest entry.
- **Bonus**: promotional credit; used after Deposit for contest entry (user-specified order: Deposit then Bonus, per request — implemented as deposit+bonus first, then winning).
- **Winning**: contest winnings and rewards; can only be used for joining after Deposit+Bonus are insufficient and is the only balance withdrawable.
- Join ordering: Debit sequence for entry fees: Deposit -> Bonus -> Winning (only if needed). This is enforced transactionally.
- Platform commission: admin-configurable per contest; commission deducted from prize pool upon payout.
- Refund on cancel: automatic full refund credited to user Deposit bucket.

---

12. Edge cases & error handling

- Prevent oversubscription: database transaction with seat count check and unique constraint on (contest_id, user_id).
- Dedup deposits by txHash; require admin check for blockchain finality.
- Withdrawal failures: admin marks Failed -> user Winning balance restored.
- Admin mistakes: support manual reversal with audit record and reason.

---

13. Testing & acceptance criteria

- Unit tests for wallet debits/credits and join flow concurrency.
- Integration tests for deposit/withdrawal lifecycle (with mocked explorer API).
- E2E test for registration flow and contest lifecycle.
- Acceptance: deposits flow to Deposit balance on admin approval; join uses correct balance order and creates entry; admin-declared winners update Winning balances correctly and allow withdrawal requests.

---

14. Open questions / decisions needed (for future clarity)

1. Exact minimum confirmations to consider a tx final (recommended: 3 confirmations for BEP20-initial, but left to admin).
2. Platform commission default percentage and rounding rules.
3. Backup frequency & retention policy (daily with 90-day retention recommended).
4. Hosting preference (cloud provider, VPS) and domain for admin dashboard.
5. Email/SMTP settings for admin alerts.
6. Exact formatting rules for contest entry codes and invitation codes (length, entropy).

---

15. Deliverables & next steps

Deliverables for you: - This PRD in Markdown (this document). - Suggested next steps for development: prioritize data model + foundational API, build admin auth + invite flows, implement bot registration + simple contest join, implement deposit-request queue and manual approval, then withdrawals.

Acceptance for handoff to engineering: - Finalize answers to the open questions above. - Provide brand assets and admin contact details.

If you want, I can now: (a) convert this into a one-page developer checklist / backlog with prioritized tickets, (b) produce the initial database schema (DDL) for PostgreSQL, or (c) draft the exact bot message copy & microcopy for each major flow. Tell me which of these you want next and I will deliver it as Markdown.