Room Style CTF - Plan, Strategise, Win

Purpose: This document outlines a feature-complete design for a gamified Capture-The-Flag (CTF) web platform comprising five sequential rooms, team-based gameplay, real-time monitoring, purchasable perks/tools, and attack/defend/invest mechanics (red-team/blue-team style). It is intended for web dev teams building the site.

1. Executive Summary

Build a responsive web-based CTF platform where all teams start in Room 1 simultaneously. Teams solve puzzles (flags) to progress to the next room. Each team begins with a fixed number of points, which they can spend to unlock one-time clues, buy perks or tools, or choose strategic actions: **Attack**, **Defend**, or **Invest**. Organisers monitor progress and can intervene. The platform supports live leaderboards, team management, secure flag submission, a role-based admin panel, and real-time updates.

2. Key Concepts & Rules

- **Rooms:** 5 sequential rooms (Room 1 → Room 5). Teams must clear the current room before moving to the next one.
- **Teams:** Groups of 1..N participants (configurable). The team captain can make purchases and decide actions. Members join via an invite link or code.
- **Points:** Every team starts with a fixed points pool (configurable). Points are currency for clues/perks/tools and actions.
- **Perks & Tools:** One-time purchase items (e.g., extra time, hint reveal, auto-validate, trace blocker). Tools are consumables.
- **Clues:** One-time unlock per clue. Each room has a set of clues that can be bought with points. Clue purchases are recorded.
- **Actions:** Each round (configurable cadence or event-based), teams may choose to Attack, Defend, or Invest — each action consumes points and has consequences against other teams or self.
- **No Flag Sharing:** sharing is prohibited and enforceable by monitoring and logs.
- **Organisers:** Admins can monitor rooms, adjust team progress, add/disable rooms, and push emergency messages.

3. Primary Features (Functional Requirements)

3.1 Authentication & Authorisation

- Sign up with email + password, optionally OAuth (Google/GitHub). Email verification is mandatory. (OTP)
- Login, password reset flows.
- Roles: `player`, `team_captain`, `admin`, `organiser`.
- Session management, JWT or server sessions with refresh tokens.

3.2 Team Formation

- Create team (name, description, capacity). Team creator becomes captain.
- Invite members via code/link OR allow open join until capacity reached.
- Leave team / transfer captain role / disband.
- Team profile page (members, points balance, history, purchases).

3.3 Room & Puzzle Management

- Each room has multiple puzzles (could be a single puzzle per room or multiple puzzles to clear).
- Puzzle types: static flag, interactive challenge (web exploitation, crypto, stego), or puzzle question.
- Flag submission UI per room with secure server-side validation.
- Per-room clue list with price and short description.
- Progression gate: team moves to the next room once the criteria are satisfied.

3.4 Shop: Perks & Tools

- Catalogue page showing purchasable perks and tools with descriptions, costs, and one-time-use labels.
- Purchase flow: captain confirms and spends points; item added to team inventory.
- Inventory UI for the team showing unused/used items and expiration, if any.

3.5 Attack / Defend / Invest Mechanics

- **Attack:** spend X points to sabotage another team (temporary delay in flag submission). Must target a specific team; attack success probability may be influenced by the defender's perks. There must be an option called “attack” in the leaderboard beside each team.
  - After every attack, the team gets an immunity power for 3mins and after that it can be attacked again.
- **Defend:** spend Y points to get protection against attacks for a time window or room attempts

- **Invest:** Invest the points to unlock clues, and there would be a challenge in every room, which can be unlocked with X points. After solving, you will get 2X of what you invested.
- All these actions should be auditable and shown in a timeline accessible to admins (but not full details to other teams to avoid meta-sabotage exposure).

3.6 Leaderboard & Scoring

- Real-time leaderboard (teams sorted by score, rooms cleared, time-to-clear tie-breakers) along with an option called "attack".
- Score components: leftover points, attack/defend achievements, and solving CTFs.
- Historical leaderboard (per event) and live leaderboard.
- Filters: by room, by points, by college/organisation (if multi-institute).

3.7 Real-time Monitoring & Live Update

- Use WebSockets (Socket.IO) or server-sent events for live updates: leaderboard, room statuses, and admin notifications.
- Organisers can view live dashboards with team positions, recent actions, purchased perks, and flag submission timestamps.
- Option to broadcast messages to all teams or specific rooms.

3.8 Admin Panel

- Manage users, teams, rooms, puzzles, flags, and perks.
- Override team progress, refund points, disable suspicious teams, and view logs.
- Spectator mode to view a specific room's activity (without revealing flags/clues in clear).

3.9 Compliance & Security

- Strong input validation and server-side flag validation (never store flags in plaintext in client code).
- Rate-limiting on flag submissions and action requests to reduce brute forcing.
- Audit logs for purchases, attacks, defence activations, and flag submissions.
- Data encryption in transit (HTTPS/TLS) and at rest for critical secrets.

4. User Flows

4.1 New User (PLAYER)

1. Onboarding: Sign up → Email verification → Create or join a team.
2. The captain sets the team name & invites members. (Either open or code-based entry)

3. Short rules modal displays (points, clue system, no flag sharing). {Details will be mentioned later}
4. Team enters Room 1 and begins solving puzzles.

4.2 Purchase Clue

1. Captain opens shop → selects clue for current room → confirms purchase.
2. Points deducted; clue unlocked and visible to the team in the room UI.
3. Clue marked used and cannot be repurchased.

4.3 Attack Flow

1. Captain selects the target team and chooses an attack perk → confirm.
2. The backend validates the cost and applies the effect (e.g., adding a cooldown to the target's submissions).
3. The attack result is pushed to both teams (with limited information to target, to avoid revealing the exact cause) and is logged for admins.

4.4 Room Progression

1. Team submits flag → server validates → if correct: assign points.
2. When enough points are gained, the users can choose to move to the next room. Server validates and deduces points to unlock the next room; update leaderboard.
3. If incorrect: increment failed attempt counter and possibly show hint-suggestion clue perk option.

5. Data Model (Suggested Tables)

Using relational DB (Postgres recommended). Primary keys as `id` UUID.

- `users` (id, email, password_hash, name, role, created_at)
- `teams` (id, name, captain_user_id, capacity, points_balance, created_at, shield_active(bool))
- `team_members` (id, team_id, user_id, joined_at, role)
- `rooms` (id, name, order_index, description, is_active, is_challenge)
- `puzzles` (id, room_id, title, type, description, flag_hash, points_reward, is_active)
- `clues` (id, puzzle_id, text, cost, is_one_time)
- `perks` (id, name, description, cost, effect_json, is_one_time)
- `purchases` (id, team_id, perk_id, purchased_at, used_at, metadata)
- `Attack` (id, attacker_team_id, target_team_id, started_at, ends_at, status (active/expired/blocked))

- `actions` (id, team_id, action_type, target_team_id, cost, result_json, created_at)
- `submissions` (id, team_id, puzzle_id, submitted_flag_hash, is_correct, submission_time, created_at, ip_address)
- `leaderboard_snapshots` (id, event_id, team_id, score, room_index, created_at)
- `audit_logs` (id, (actor)user_id, action, details_json, created_at)

6. API Endpoints (Representative)

Auth

- `POST /api/auth/signup` — create account
- `POST /api/auth/login` — login
- `POST /api/auth/verify-email` — verify

Teams

- `POST /api/teams` — create team
- `POST /api/teams/:id/invite` — create invite
- `POST /api/teams/:id/join` — join via code
- `GET /api/teams/:id` — team profile

Game

- `GET /api/rooms/:id` — retrieve room details & available puzzles/clues
- `POST /api/puzzles/:id/submit` — submit flag
- `POST /api/clues/:id/buy` — buy clue
- `POST /api/perks/:id/buy` — buy perk
- `POST /api/actions` — perform attack/defend/invest
- `GET /api/leaderboard` — live leaderboard

Admin

- `POST /api/admin/rooms/:id/override-progress` - move team forward/backward
- `GET /api/admin/logs` - fetch audit logs
- `POST /api/admin/rooms` - To access and edit questions, answers and clues.
- `GET /api/admin/team` - Get team and users' details along with position in leaderboard.

All write endpoints must be protected by authentication and rate-limited.

7. Real-time Architecture

- **Realtime server**: Node.js with Socket.IO or Phoenix Channels (Elixir) for scale.
- Channels:
  - `leaderboard` — broadcasts rank updates.
  - `team:{teamId}` — team-specific events (clue unlocked, purchase updates, private messages from admin).
  - `room:{roomId}` — room-wide broadcasts.
  - `admin` — admin dashboard events.
- Events are emitted when: flag validated, purchase made, action resolved, leaderboard change.
- Use a message queue (Redis Pub/Sub or RabbitMQ) to scale across instances.

8. UI / UX Guidelines

- **Homepage:** Event name, countdown timer, quick rules, Sign up / Login.
- **Dashboard:** Team status, current room, active puzzle, remaining points, inventory, action buttons.
- **Room Screen:** Puzzle description, file upload (if needed), flag submission box, buy clue button, team chat (optional, moderated), and recent activity feed for the team.
- **Shop:** Filter by perks, show costs and "one-time" badges.
- **Leaderboard:** prominently shown with real-time updates; small badges for teams who are currently under attack/defence.
- **Admin Panel:** compact monitors: team list with room index, points, recent actions, and quick controls (move team, refund, ban).

9. Security Considerations

- **Flag Handling:** store hashed flag values (use HMAC with server secret) and validate server-side. Never embed flags or verification logic in client code.
- **Rate limit:** per-team and per-IP limit for submissions and actions.
- **Anti-cheat:** detect suspicious patterns (multiple teams using the same IPs, repeated exact flag attempts across teams); flag for human review.
- **Access control:** team-only endpoints must verify membership.
- **Logging & Monitoring:** detailed logging of buys, attacks, and submissions.

10. Admin Tools & Monitoring

- **Live Dashboard** with sortable team list and quick actions.
- **Logs Viewer** with filters (time range, team, event type).
- **Manual Override** to move teams, refund points, or mark puzzle solved.
- **Broadcast** interface to send messages to all teams or specific rooms.
- **Anti-cheat flagger**: automation rules.

11. Edge case considerations

- **Simultaneous purchases/attacks:** use DB transactions and checks to avoid negative balances or simultaneous unlocking race conditions.
- **Attack during instant shield purchase:** atomic check at initiation time; if shield activates between UI and server, server must reject attack and inform attacker.
- **Team leader leaves:** allow auto-transfer or require manual transfer before leaving. If the leader is disconnected, provide a grace period.