Code Royale — Beta Roadmap & Full Feature Plan

> Complete, saveable plan for building, releasing, and scaling **Code Royale** (competitive coding PvP). Designed for a first polished beta (1v1) and a roadmap for all features.

---

1. Project overview

Goal: Build a competitive, real-time coding battle web app where players fight 1v1 (beta) to solve coding problems fastest and correctly. Play modes later include 4-player battles, bots, ranked ladders, and tournaments.

Primary constraints for beta: - Start with **1v1 matches only** (live matchmaking and friend invites) - Essential features: auth, matchmaking, live code editor, judge integration, basic leaderboard, minimal UX polish - Use **Next.js** (frontend + API routes) and **Supabase** (Postgres, Auth, Realtime) - Use **Judge0** (or similar) for safe code execution

---

2. Beta (MVP) feature list — must-have

Core gameplay

- 1v1 **Live Match** flow (matchmaking or friend invite)
- Code editor in-browser (Monaco or CodeMirror) with language selection
- Timer for matches; configurable time limit for each match (MVP: 5/10/20 minutes)
- Automatic judging via Judge0 of submissions against test cases
- Score calculation: correctness + speed (first to pass all tests wins)
- Minimal UI: lobby, match screen, results screen (who won, stats)

Account & social

- Sign up / Login (Supabase Auth) — email + OAuth (optional)
- Basic profile: username, avatar, rating (ELO-ish), win/loss
- Friend system: send/accept friend requests, friend list, invite friend to 1v1

Data & persistence

- Store users, matches, match results, question metadata, ratings, leaderboards
- Seed question bank with ~50–150 curated problems for beta across difficulties

Admin & moderation (minimal)

- Admin view to add/edit/remove questions and test cases
- Basic anti-abuse: limit submissions per second; simple rate-limits

Dev ops

- Deploy frontend/backends to Vercel (Next.js)
- Supabase project for DB & realtime channels
- CI: GitHub actions for linting and tests

---

3. Full product feature backlog (post-beta)

Short term (post-beta) - Ranked ladder (divisions/leagues, ELO/TrueSkill) - Practice Arena: single-player practice with unlimited tries - Bot opponent mode (scripted bots + adjustable difficulty) - 4-player free-for-all match mode (last to solve loses or points-based) - Hints system (costs rating or in-game currency) - Match spectating / replays (record submissions timeline)

Medium term - Tournaments & scheduled events - Chat & messages (outside matches), but no chat during match - In-match anti-cheat: detect paste patterns, suspiciously instant passes - AI-detection (separate feature): flag possible AI-generated code for review - Economy: cosmetics, skins, avatars

Long term - Esports features: broadcasting, streaming integration, ladder seasons - Mobile app wrappers (React Native / Expo) - Advanced analytics and replayable diff viewers

---

4. Tech stack & architecture

Frontend: Next.js (App Router) + Tailwind CSS + Monaco Editor (or CodeMirror)

Realtime: Supabase Realtime (Postgres replication via realtime channels) OR Socket.io if you prefer custom server.

Database & Auth: Supabase (Postgres + Auth + Policies)

Code judge: Judge0 (hosted or paid plan); server-side API route calls Judge0

Hosting / Deploy: Vercel (Next.js), Supabase for DB

Optional: Redis for matchmaking queue or cache (later)

High-level flow:

```
Client (Next.js) <--> Supabase (Auth + DB + Realtime)
Client (via server API) --> Next.js API route --> Judge0 API
Match events pushed via Supabase realtime channels to clients
```

---

5. Database schema (recommended tables)

users

- id (uuid, PK)
- username (text, unique)
- display_name
- avatar_url
- rating (int, default 1200)
- wins (int)
- losses (int)
- created_at (timestamp)

friends

- id (uuid)
- user_id
- friend_id
- status (pending, accepted)
- created_at

questions

- id (uuid)
- title
- slug
- description (md or plaintext)
- difficulty (enum: easy/medium/hard)
- languages_allowed (jsonb)
- testcases (jsonb) — array of { input, expected_output }
- created_by (uuid admin)
- created_at

matches

- id (uuid)
- mode (1v1, 4p, bot)
- status (queued, running, finished)
- time_limit_seconds
- started_at
- finished_at
- question_id
- created_by (user id who initiated)

match_players

- id
- match_id
- user_id (or 'bot_x')
- submission_history (jsonb) — array of { timestamp, code, passed_count, total_count }
- final_result (win/lose/draw)
- score

leaderboard (materialized view or cache)

- user_id
- rating
- rank

---

6. API routes & realtime channels (Next.js app/api)

REST API routes (examples)

- `POST /api/auth/callback` (if using custom flows)
- `GET /api/questions/random?difficulty=easy` (get question)
- `POST /api/match/create` (create match / invite)
- `POST /api/match/join` (join existing match)
- `POST /api/match/submit` (submit code -> call Judge0 -> store result)
- `GET /api/leaderboard/top?limit=50`
- `POST /api/friend/request` (send friend request)

Realtime events via Supabase channels (or socket rooms)

- `match:<match_id>` channel
- Events: `match_started`, `player_submitted`, `player_passed`, `match_ended`, `player_left`
- `users:<user_id>` channel
- Events: `friend_request`, `friend_online`

---

7. Real-time judging flow (detailed)

1. Client submits code (via POST `/api/match/submit`) with fields: match_id, user_id, language, source_code
2. Next.js API route authenticates caller (Supabase token check)
3. Server composes request for Judge0 with multiple testcases (in parallel or sequentially)
4. Send to Judge0; wait for response (or poll if async). For beta, use wait=true so single request returns verdict.
5. Parse Judge0 response; compute `passed_count / total_count`
6. Save submission snapshot to `match_players.submission_history`
7. Emit realtime event to `match:<match_id>` with result (so opponent sees it)
8. Check match win conditions (all tests passed or time expired) -> finalize match
9. Update ratings and leaderboard via dedicated function

Notes: - To avoid DoS, throttle submissions per user (e.g., 1 submission per 2s) and require a short debounce client-side. - For speed, send only input cases that matter (a mix of sample + hidden tests); runtime must be limited.

---

8. Rating system (simple start)

- Start with **ELO** or simplified ELO-like points:

- Default rating = 1200. Win against higher rated grants more points.
- Use K-factor (e.g., 32) tuned later.
- Store rating changes per match for auditing.

---

9. Question bank & seeding (beta)

- Seed initial bank with **50–150** problems:
- 30 easy, 15 medium, 5 hard.
- Each problem must have at least 5 testcases (2 sample, 3 hidden).
- Sources: write original problems inspired by common algorithms (arrays, strings, sorting, sliding window, hashing). Avoid copying copyrighted LeetCode text.
- Create an admin UI for adding/editing testcases and viewing submissions.

---

10. Bots & practice mode (basic)

Bots: - Version 1: scripted bots with deterministic speeds (e.g., will pass easy in 12s, medium in 40s) — simulate typing + submit events - Version 2: implement code snippets that actually solve only easiest tasks (for more realism)

Practice mode: - Allow users to pick a question and submit without affecting rating

---

11. Security, anti-cheat & AI detection (planning)

- Enforce CORS and require Supabase JWT for API calls
- Validate user identity server-side before accepting submissions
- Rate-limit submissions per match/per user
- Do not expose hidden testcases to clients
- Logging: record IPs, user agent, submission times for suspicion review

AI detection (later): - Collect suspicious samples and use heuristics (e.g., very short solve times, similar code across users) - Optionally run a lightweight similarity check (Levenshtein/diff) to detect near-identical code - For stronger detection, integrate third-party AI-detection services (paywalled)

---

12. Testing & QA

- Unit tests for rating calculations, match resolution logic
- Integration tests for `/api/match/submit` talking to Judge0 (use a mock during CI)
- End-to-end tests for full match flows using Playwright or Cypress
- Load tests for matchmaking path and Judge0 timeouts

---

13. Deployment & monitoring

- Deploy Next.js to Vercel (branch previews for PRs)

- Supabase hosted DB (production project)
- Use Sentry or Vercel analytics for crash/telemetry
- Monitoring alerts for error rates and Judge0 latency

---

14. Launch (beta) checklist

- [ ] 1v1 live match fully functional
- [ ] Auth + profiles + friend invites
- [ ] Question bank seeded (50+) and admin UI
- [ ] Judge0 integration robust and time-limited
- [ ] Leaderboard & rating updates
- [ ] Rate-limiting & basic security policies
- [ ] Tests & CI passing
- [ ] Deployable to Vercel + Supabase production
- [ ] Privacy & Terms pages (basic)

---

15. Suggested milestones (priority)

- **Milestone A (Week: build quick MVP)**
- Authentication + profile
- Single-player judge test page (test judge0 integration)
- 1v1 matchmaking + simple match flow

- Basic UI for matches and results

- **Milestone B**

- Friends system + friend invite to match
- Persistent matches stored in DB

- Rating calculation and leaderboard

- **Milestone C**

- Bots + practice arena

- Admin UI + question editor

- **Milestone D**

- Polish, tests, deploy, beta release

(Adjust weeks per your schedule.)

---

16. Next immediate actions you can copy into tasks

1. Create Supabase project & set up `users`, `questions`, `matches` tables

2. Initialize Next.js app with Tailwind and Monaco editor
3. Implement `/api/judge` route that talks to Judge0 (mock locally first)
4. Build single-player judge page to prove end-to-end judge flow
5. Implement simple matchmaking logic using Supabase realtime channels
6. Implement match UI and realtime updates
7. Seed 50 starter questions
8. Test and deploy to Vercel + Supabase

---

17. Notes on costs

- Supabase free tier is sufficient for beta. Judge0 may require a paid plan if you exceed rate limits. Vercel free tier fine for early users.
- Monitor Judge0 usage and consider hosting your own Judge0 or switching to paid plan if usage grows.

---

18. Final advice

- Keep the beta **laser-focused** on the 1v1 experience and polish that flow until it's fun and reliable. Postpone chat, complicated AI detection, and tournaments until you have reliable judge and matchmaking.
- Make the question authoring/admin UI early — it saves time.
- Record metrics: match duration, judge latency, submission counts — these matter when you scale.

If you want, I'll also: - generate the exact **SQL schema** `CREATE TABLE` statements for Supabase, - or create the **Next.js API route** example for `/api/match/submit` integrating Judge0, - or scaffold the **navbar + homepage** components.

Tell me which of those you want next and I'll generate it and save a copy for you.