Technical Documentation — Qiyam (urqiyam)

> Purpose: concise, surgical breakdown of the tech stack, architecture, deployment, security assumptions and developer runbook.

---

1 — High level

Qiyam is a Next.js (App Router) PWA that uses Supabase (Postgres + Auth) as its backend and OpenAI for idea generation. The UI is built with React + Tailwind CSS and uses Framer Motion for animations. The app is Vercel-ready and targets Node.js 18.x runtime.

Primary goals - Server-side AI generation (OpenAI) with user-specific personalization stored in Supabase. - Auth via Supabase Auth, client uses anon key for public operations and server functions use secured keys. - Database-driven flow: quiz → ideas generation → swipe → persistent ideas/ventures.

---

2 — Tech stack (precise)

- **Framework / UI**: Next.js 16 (App Router), React 19 (server components + client components supported)
- **Styling**: Tailwind CSS v4 (utility-first styling)
- **Animations**: Framer Motion
- **Backend / DB**: Supabase (PostgreSQL), Supabase Auth
- **AI**: OpenAI (recommended model: `gpt-4o-mini` as set in README/ENV)
- **Runtime**: Node.js >= 18.17 (dev/prd)
- **Deployment / Hosting**: Vercel (recommended) — uses Next.js first-class features (edge/ISR/SSG as applicable)
- **Language / Tooling**: TypeScript (tsconfig.json present), ESLint (eslint.config.mjs), PostCSS (postcss.config.mjs)

Files & folders (source-of-truth in repo) - `app/` — Next.js App Router routes, pages and API routes - `app/api/` — server API endpoints (AI generation, integrations) - `app/quiz`, `app/ideas`, `app/auth` etc — feature routes - `components/` — reusable React components - `lib/` — utility modules and datasets - `utils/supabase/` — Supabase client initialization (client and server clients) - `schema.sql` — canonical DB schema (run in Supabase SQL editor) - `middleware.ts` — session and route protection; Next.js middleware used for auth/session enforcement - `next.config.ts` — Next.js runtime configuration

---

3 — Environment variables (must be present)

Put these in `.env.local` (local dev) and in Vercel/GitHub secrets for production:

- `NEXT_PUBLIC_SUPABASE_URL` — Supabase project URL (public safe)
- `NEXT_PUBLIC_SUPABASE_ANON_KEY` — Supabase anon (client) key (public)
- `SUPABASE_SERVICE_ROLE_KEY` — Supabase service_role (server-only) **DO NOT** expose to client
- `OPENAI_API_KEY` — OpenAI API key (server-side only)
- `OPENAI_MODEL` — model to use (e.g. `gpt-4o-mini`)
- `NEXT_PUBLIC_SITE_URL` — site public URL

Notes: Any key prefixed with `NEXT_PUBLIC_` is embedded in client bundles — never prefix `SUPABASE_SERVICE_ROLE_KEY` or `OPENAI_API_KEY` with `NEXT_PUBLIC_`.

---

4 — Database (schema summary)

Run `schema.sql` in the Supabase SQL editor. Key tables (as described in README):

- `users` — user profile, auth metadata, `dna_json` (user quiz results), timestamps
- `ideas` — generated ideas, `author_user_id`, `idea_json` (structure for descriptions, metrics), `status` (generated / rejected / selected)
- `ventures` — persisted ventures the user has selected
- `challenges` — weekly tasks or prompts
- `scores` — numeric metrics (barakah, business viability, sustainability) — used for leaderboards or progress

Implementation detail: Use `jsonb` for structured blobs (`dna_json`, `idea_json`) and index using GIN where appropriate. Add indexes on `user_id`, `created_at`, `status` for query performance.

Security: Make use of Supabase Row Level Security (RLS) — README explicitly notes RLS policies exist. Ensure RLS policies grant read/write to authenticated users for their own rows only, and write limited server role for background processes.

---

5 — Authentication & session flow

- User signs up / signs in via Supabase Auth (email/password or magic link). The client uses `NEXT_PUBLIC_SUPABASE_ANON_KEY` to talk to Supabase Auth.
- Server-side protected routes (API routes that should be authenticated) must verify the Supabase JWT or call Supabase with `SUPABASE_SERVICE_ROLE_KEY` when performing privileged operations.
- `middleware.ts` sits at the edge to redirect unauthenticated requests from protected pages or to attach session info to requests.

Best practice: Validate JWT server-side in API routes (not rely on client-only checks). Use `getUser` helpers from Supabase server SDK in server/edge functions.

6 — OpenAI integration (idea generation flow)

- Idea generation should run in a server-only environment (API route under `app/api/`), never in client code.
- The server code reads `OPENAI_API_KEY` from the environment, constructs a prompt (likely using the user `dna_json`), and requests `OPENAI_MODEL` (e.g. `gpt-4o-mini`).
- Results are parsed, normalized, persisted to `ideas` table, and returned to the client.

Cost & rate control: - Gate calls behind a job queue or a rate limiter per user (e.g., 1 generation per X seconds) to avoid runaway costs. - Cache model outputs for repeated requests with same inputs (hash prompt → results).

Failure modes: Handle OpenAI timeouts and partial results. Implement idempotency keys for retries.

---

7 — API & server considerations

- Keep API surface minimal and focused: `POST /api/generate-ideas`, `GET /api/ideas`, `POST /api/ideas/:id/pick` (conceptual — actual route names live under `app/api/`).
- Validate inputs with Zod or similar before sending to OpenAI / DB.
- Use response size limits and sanitize model output before persisting (strip unsafe HTML or scripts).

---

8 — Caching & client data fetching

- Use SWR (or React Query) for client-side data fetching and optimistic updates.
- For frequently-read public pages (landing page), use Next.js caching headers or ISR.
- For user-specific dashboards, use server components and fetch server-side when possible to reduce client surface.

---

9 — Dev setup & commands

```
# 1. Install
npm install

# 2. Local env
cp .env.local.example .env.local
# fill in envs: NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY,
OPENAI_API_KEY, OPENAI_MODEL, NEXT_PUBLIC_SITE_URL

# 3. Run DB schema in Supabase SQL Editor (copy content of schema.sql)
```

```
# 4. Start dev server
npm run dev

# 5. Build (production)
npm run build
npm run start
```

Lint & type-check - `npm run lint` (assuming ESLint configured) - `npm run typecheck` or `tsc --noEmit`

---

10 — Deployment (Vercel)

Steps 1. Push `main` to GitHub 2. Import project into Vercel 3. Add environment variables in Vercel (server-only vars as `OPENAI_API_KEY`, `SUPABASE_SERVICE_ROLE_KEY`) 4. Set build command (default Next.js) and output settings if needed 5. Deploy

Edge considerations - If using Next.js edge functions (or middleware), ensure `OPENAI_API_KEY` is not leaked — do not call OpenAI from true edge runtime unless the provider supports secure environment variables there.

---

11 — Observability, logging & errors

- Server: Use structured logs when calling OpenAI and Supabase (request id, user id, prompt hash, latency, token usage if available).
- Production: Consider Sentry (error tracking) and a lightweight analytics tool (Plausible/Heap/Amplitude) for product usage.
- DB: Monitor Supabase query performance and set up alerts for slow queries and high connection counts.

---

12 — Security checklist

- Do NOT expose `OPENAI_API_KEY` or `SUPABASE_SERVICE_ROLE_KEY` to client bundles.
- Enforce RLS on tables and test policy coverage with a non-privileged user.
- Rate-limit OpenAI endpoints by user ID and IP.
- Sanitize and validate all user inputs (Zod or Joi).
- Use HTTPS in production (Vercel manages this by default).

---

13 — Testing recommendations

- Unit: jest + ts-jest for pure logic (prompt builders, parsers)

- Integration: Playwright or Cypress for end-to-end flows (signup → quiz → idea generation → swipe)
- Contract: test Supabase RLS with automated tests that attempt unauthorized access

---

14 — File map (notes)

- `app/` — App Router: server and client components, API routes
- `components/` — UI primitives & atoms
- `lib/` — shared logic (prompt templates, scoring functions)
- `utils/supabase/` — Supabase clients (client vs server instantiation)
- `schema.sql` — canonical schema to run in Supabase
- `middleware.ts` — auth/session enforcement
- `next.config.ts` — Next.js config (image domains, rewrites, experimental flags)
- `technical.md` — (this file)

---

15 — Short roadmap & hardening (suggested)

1. Add API-level rate limiting + idempotency for AI calls.
2. Add request/response audits: save prompt hashes, token counts (if provided) to track cost.
3. Add Sentry + performance monitoring.
4. Add e2e test that runs against a staging Supabase and a sandboxed OpenAI key.
5. Add CI checks: lint, typecheck, unit tests on PRs.

---

16 — What I could not fetch automatically

The repository README provides most stack details; however, the exact dependency versions from `package.json` and the full `schema.sql` content were not accessible through the web viewer at the time of writing (GitHub file viewer intermittent error). Before finalizing a production-ready checklist, confirm the following directly from the repo locally or via the raw file endpoints:

- Exact package dependency versions (in `package.json`) — lockfile (`package-lock.json` or `pnpm-lock.yaml`) if present
- Exact Supabase RLS policy definitions and `schema.sql` column types
- Exact API route filenames and their request/response contracts

---

If you want, I can: - produce a PR patch that replaces or expands the existing `technical.md` in the repo with this document, including a small `CONTRIBUTING` checklist and CI steps (I will need write access or a fork + PR flow) - extract the exact versions and schema (I attempted but GitHub file viewer returned an intermittent error — if you want, paste the `package.json` and `schema.sql` raw contents here and I will embed exact-lines and versioned recommendations).

End of technical.md