CarbonJar Technical Report

Date: 2025-10-06 Repository: CarbonJar (branch: main)

1. Executive Summary

CarbonJar is a Next.js 15 App Router application built with React 19 and TypeScript 5. It authenticates users via Clerk, persists data in PostgreSQL via Drizzle ORM, and optionally uses Upstash Redis for distributed API rate limiting. The project includes an OpenAPI 3.1 specification with a Swagger-based viewer, a modular component library, and a CI workflow covering lint, types, and tests.

2. System Overview

- Frontend/SSR: Next.js 15 (App Router), React 19
- Language: TypeScript 5 (strict)
- Styling: Tailwind CSS v4 via @tailwindcss/postcss; custom themes in `app/globals.css`
- Auth: Clerk with middleware protection and role checks
- Database: PostgreSQL (Neon serverless runtime adapter preferred)
- ORM: Drizzle ORM with first-class schema and migrations
- Caching/Rate limiting: Upstash Redis (optional) with in-memory fallback
- Testing: Jest 30, React Testing Library, jest-dom
- CI: GitHub Actions on Node 20 (lint, typecheck, test)

3. Repository Structure

- `app/` Next.js App Router pages, layouts, and API routes (`app/api/*`)
- `components/` Reusable UI components grouped by feature
- `lib/` Auth, env validation, rate limit, security helpers; database client and schema under `lib/db/*`
- `migrations/` Drizzle SQL migrations output
- `__tests__/` Unit and component tests
- `openapi.yml` OpenAPI 3.1 spec; `/api-docs` renders swagger.html
- `public/` static assets (logos, swagger.html, fonts)
- `postcss.config.mjs`, `next.config.ts`, `tsconfig.json`, `jest.config.js`

4. Runtime Configuration and Security

- Security headers (`next.config.ts`):

    - X-Frame-Options: SAMEORIGIN
    - X-Content-Type-Options: nosniff
    - Referrer-Policy: strict-origin-when-cross-origin
    - Permissions-Policy: geolocation=(), microphone=(), camera=()
- Middleware (`middleware.ts`):
    - Clerk auth for protected routes; public routes include `/`, `/about`, `/expertise`, `/trainings(.*)`, `/api/trainings(.*)`, `/api/contact`, `/api/users(.*)`, `/api/webhooks(.*)`, `/sign-in(.*)`, `/sign-up(.*)`
    - CSRF origin checks block cross-site POST/PUT/PATCH/DELETE (webhooks excluded)
    - API rate limit via Upstash (100/min sliding window) if configured
    - Strict CSP with extended img/connect/font directives
- Input sanitization: `lib/xss.ts` provides `escapeHtml`
- Env validation: `lib/env.ts` via Zod; rejects missing critical secrets

5. Environment Variables

Required:

- `DATABASE_URL` (PostgreSQL)
- `CLERK_SECRET_KEY`
- `NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY`

Optional:

- `UPSTASH_REDIS_REST_URL`, `UPSTASH_REDIS_REST_TOKEN` (distributed rate limiting)
- `CLERK_WEBHOOK_SECRET` (for `/api/webhooks/user`) — referenced in README

6. Database Schema (Drizzle ORM)

Schema file: `lib/db/schema.ts`. Key entities:

- `auth_users` (user profile, Clerk linkage via `clerk_id`, role enum, status flags)
- `carbon_topics`, `courses` (course metadata with level/status, duration/price/why)
- `course_prerequisites` (self-join on courses)
- `modules` (content, type, order, estimatedDuration)
- `assessments`, `questions`
- `certificates` (fullName, title/description, course dates, issuer info, `certificate_code`, `certificate_slug`, `pdf_url`, `certificate_hash`, revoke fields)
- `enrollments`, `training_sessions`
- `learning_analytics` (aggregate counters)
- `contactrequests` (type/status/priority, scheduling fields, linked user)
- `notifications`

Runtime clients:

- `lib/db/drizzle.ts` (Neon serverless + drizzle-orm/neon-http, lazy-initialized)
- In some routes (`app/api/trainings/route.ts`), `pg` + `drizzle-orm/node-postgres` is used with per-request connections

7. API Surface

OpenAPI: `openapi.yml` (3.1) — exposed via `/api-docs` using `public/swagger.html`.

Implemented examples:

- `GET /api/contactrequests` — lists all contact requests (ordered by `submitted_at` desc)
- `POST /api/contactrequests` — rate-limited (8/min per IP), validates enum fields, escapes strings, inserts row; returns created entity
- `GET /api/trainings` — lists all `courses`
- `POST /api/trainings` — requires Clerk auth + role in {trainer, admin}; normalizes status (trainers -> Draft); inserts course; creates a default `training_sessions` row

Patterns:

- AuthZ: `lib/auth.ts` exports `requireAuth({ roles })` to gate roles from Clerk metadata
- Rate limiting: global in middleware (Upstash) + per-route fallback via `lib/rateLimit.ts`
- Data validation: narrow `req.json()` and check enums using schema enum values

8. Frontend and Styling

- Tailwind v4 via PostCSS plugin (@tailwindcss/postcss) configured in `postcss.config.mjs`
- Global theme tokens, fonts, and UI behavior in `app/globals.css` (includes Clerk component overrides and animation/perf optimizations)
- Image domains allowed in `next.config.ts`: logos-world.net, upload.wikimedia.org, cdn.jsdelivr.net

9. Testing

- Jest (`jest.config.js`):
  - Environment: jsdom; setup: `jest.setup.js` (RTL and jest-dom)
  - Coverage from `app`, `components`, `lib`, `hooks`
  - `@/` path alias to project root

- Examples:
  - `__tests__/components/certificate-card.test.tsx` — verifies public credential link and LinkedIn Add-to-Profile URL params
  - `__tests__/api/contact.test.ts` — smoke test scaffold

10. Developer Experience and CI

- `package.json` scripts:
  - dev: `next dev --turbopack`
  - build/start, lint/lint:fix, typecheck, format/format:check, test variants
  - `prepare`: husky install; lint-staged configured for per-file lint + Prettier
- GitHub Actions: `.github/workflows/ci.yml` with Node 20; steps: checkout → install → lint → typecheck → test

11. Local Development

- Install: `npm ci`
- Run dev: `npm run dev` (http://localhost:3000)
- API docs: http://localhost:3000/api-docs (renders `openapi.yml`)
- Env: create `.env.local` with required variables (see §5). For Upstash-based rate limiting, set Redis URL/token.

12. Deployment Considerations

- Set `DATABASE_URL`, Clerk keys, and (optionally) Upstash envs
- Ensure Next.js image domains are permitted in deployment
- Middleware relies on `x-forwarded-for`/`x-real-ip` headers for rate limiting — confirm your proxy/CDN forwards these
- For serverless Postgres (Neon), use `@neondatabase/serverless` friendly environments

13. Risks and Improvements

- Mixed DB access adapters (neon-http vs pg client) across routes — standardize to a single adapter for consistency and connection handling
- Add schema-level validation (Zod or valibot) for request bodies beyond enum checks
- Expand OpenAPI spec to cover all implemented endpoints and error schemas
- Add e2e tests (Playwright) for critical flows (auth, trainings CRUD, contact form)
- Consider adding caching headers or ISR for public pages
- Add `.env.example` checked into repo (README references it but file not present)

14. Appendix: File References

- Config: `next.config.ts`, `tsconfig.json`, `postcss.config.mjs`
- Middleware: `middleware.ts`
- Env/Security: `lib/env.ts`, `lib/security.ts`, `lib/xss.ts`, `lib/rateLimit.ts`
- Auth helpers: `lib/auth.ts`
- DB: `lib/db/schema.ts`, `lib/db/drizzle.ts`, `drizzle.config.ts`, `migrations/*`
- APIs: `app/api/contactrequests/route.ts`, `app/api/trainings/route.ts`
- Tests: `__tests__/*`
- Docs: `openapi.yml`, `public/swagger.html`

Generated on 2025-10-06T18:41:46.592Z