Developer Toolkit

Independent project · 2026 · Solo · Public repository · Live

Twenty everyday developer tools in one place, from diffing .env files to decoding JWTs and pulling tables out of PDFs, all running in the browser.

Links: Live demo · Repository (kagem)

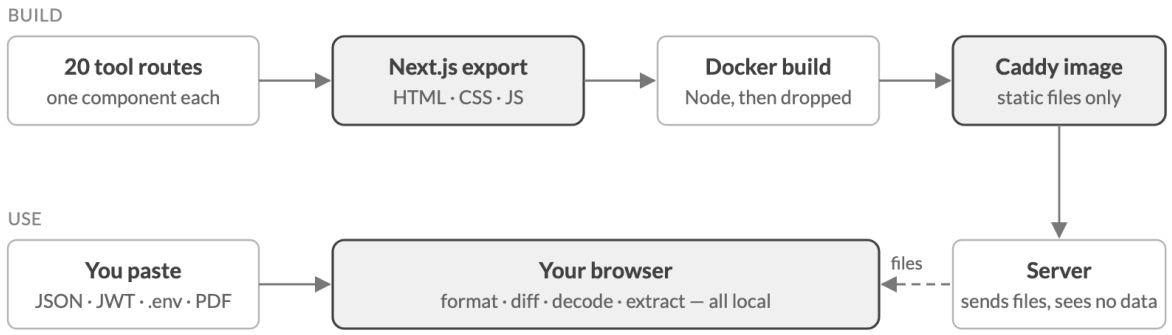
Why

Daily backend work means pasting JSON, SQL, JWTs and .env files into whichever online formatter comes up first, often with real tokens and config inside. I wanted the tools I reach for every day in one place, with nothing sent to a server, usable from any device on my network.

What it does

- Compares text line by line, and two .env files key by key.
- Formats and converts JSON, YAML, .env, XML, SQL and GraphQL, with tree and schema views.
- Extracts tables from text-based PDFs into CSV.
- Generates UUIDs (v4, v7), hashes and HMACs, random strings and placeholder text.
- Decodes JWTs, parses URLs and cron expressions, tests regexes, and converts timestamps, colours and number bases.

Key flow



1. Each tool is its own route with one client component; adding a tool is a page, a component and one entry in the tool list.
2. The build is a static export: plain HTML, CSS and JavaScript, with no server code.
3. A Docker build compiles it with Node, then copies only the output into a Caddy image that serves the files.
4. Everything a user pastes is processed in the browser and never leaves it.

Key decisions

Decision	Why / trade-off
Static export served by Caddy	No backend means no data leaves the browser, and the final image has no Node and no node_modules. The cost: no feature can rely on a server.

crypto-js for HMAC instead of Web Crypto	Web Crypto only works over HTTPS, and the toolkit is also used over plain HTTP on the LAN. The trade-off is one more dependency.
Clipboard fallback over HTTP	The async clipboard API also needs HTTPS, so copy falls back to the older command and paste asks for Ctrl+V.
PDF tables from text geometry	Rows are grouped by baseline, cells by horizontal gap, and columns per block of rows. No OCR or ruling lines, so merged or wrapped cells can split.
Undo for whole operations	In the JSON formatter, Format, Minify and Sort keys sit on their own undo stack, while the editor still undoes each keystroke.

Non-goals

- Accounts, sync or server-side storage: nothing is saved outside the browser.
- Scanned PDFs: tables are read from the text layer, not from images.

Outcome

- Live at kagem.yesyuk.com, with 20 tools across comparators, formatters, generators and encoders.
- Shipped as a small static container: Node only at build time, Caddy at run time.
- Lesson: browser APIs such as Web Crypto and the async clipboard need a secure context; a tool meant for the LAN has to plan for plain HTTP from the start.

Code pointers

- src/components/pdfTable.ts: table extraction from PDF text positions.
- src/components/JsonFormatter.tsx: tree view, schema inference and the operation undo stack.
- Dockerfile: the two-stage build into a Caddy image.

Stack

Next.js · React · TypeScript · Tailwind CSS · Ace Editor · pdf.js · Caddy · Docker