Project information

Created on
February 27, 2026

Name	Last update
backend	20 minutes ago
caddy	2 hours ago
frontend	20 minutes ago
scripts	1 hour ago
.dockerignore	3 hours ago
.env.example	1 hour ago
.gitignore	15 hours ago
.prettierignore	16 hours ago
.prettierrc.json	16 hours ago
CLAUDE.md	20 minutes ago
README.md	14 minutes ago
bun.lock	1 hour ago
docker-compose.dev.yml	1 month ago
docker-compose.yml	41 minutes ago
eslint.config.js	16 hours ago
justfile	1 hour ago
package.json	20 minutes ago
slides.md	20 minutes ago

README.md

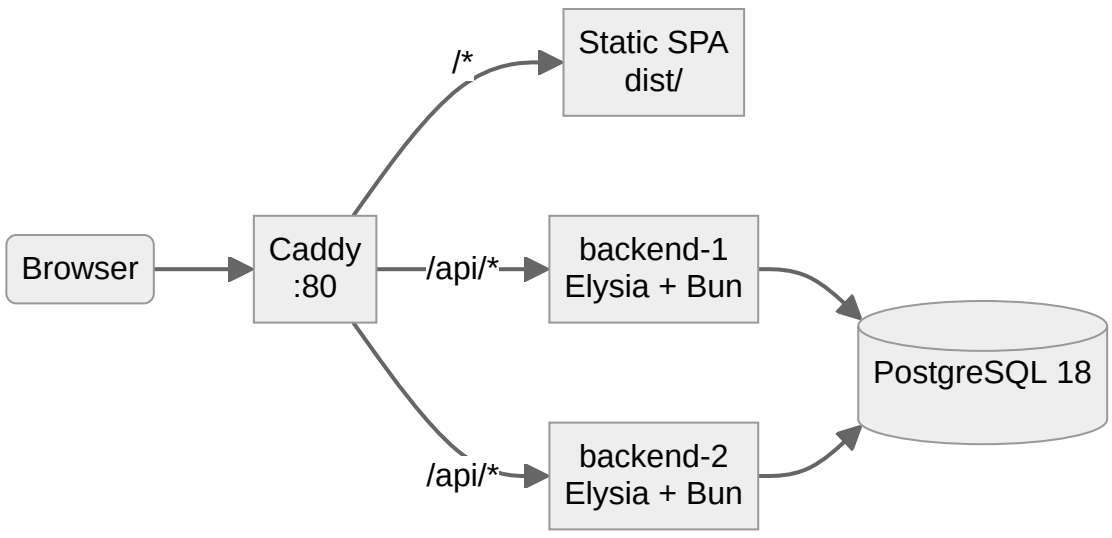
paste-it

A small Pastebin-style app: create, share, and manage text/code snippets. Anyone with a link can view a paste; creating and deleting requires an account.

Stack

- **Frontend** — Svelte 5 (runes) + Vite + sv-router, shipped as a static SPA
- **Backend** — Elysia on Bun, two instances behind a load balancer
- **Auth** — JWT (15-min access + 7-d refresh, rotated with reuse detection) in `httpOnly` cookies via jose; argon2id passwords (`Bun.password`)
- **Edge** — single Caddy 2 container that serves the SPA *and* round-robins `/api/*` to the backends
- **Database** — PostgreSQL 18

Architecture



Four containers on one Docker network; Caddy is the only host-exposed service (`:80`). It serves the SPA `dist/` for non-`/api` paths and round-robins `/api/*` between the two backend instances, with active health checks for instant failover if a backend dies. Demos: `just lb-rr` (watch round-robin), `just lb-failover` (watch failover).

Getting started

Install Bun, Docker (with Compose v2), and just, then run once:

```
just setup    # creates .env files and generates JWT_SECRET
```

From there, pick one:

Production-like stack

```
just up       # full stack → http://localhost
just down     # tear down (incl. db volume)
```

Dev with hot reload

```
just dev      # postgres + backend + Vite, all in one terminal → http://localhost:5173
```

Ctrl+C stops everything (including postgres). Vite proxies `/api` to the backend on `:3000`.

Design notes

Stack — what and why

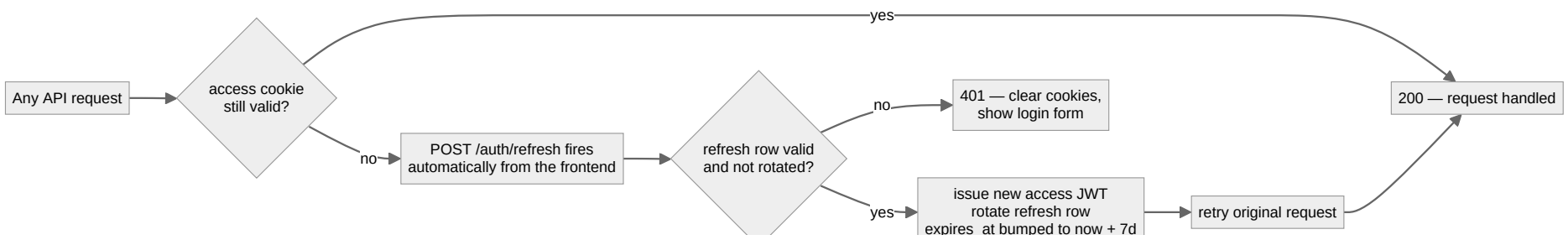
Layer	Choice	Why
Runtime	Bun + Elysia	one runtime, native TS, ~3× Node Express
Frontend	Svelte 5 (runes) + sv-router	code-based routing in one config; pure SPA, no SSR overhead
ORM	Drizzle	type-safe SQL, zero-runtime, schema-as-code
Auth	jose + Bun.password	stateless JWT (HS256, `httpOnly` cookie) + argon2id
Edge	Caddy 2	proxy *and* static server in one container
DB	PostgreSQL 18	familiar, FK-enforced, drizzle-kit migrations

Auth — OWASP mitigations

Concern	Mitigation
`alg=none` smuggling	`jwtVerify` with explicit `algorithms: ["HS256"]`
XSS exfiltration	both tokens in `httpOnly` cookies — unreachable from JS
CSRF	`SameSite=Lax` + browser-required `Origin` header (CORS)

Concern	Mitigation
Token revocation	15-min access + 7-d refresh with rotation; reuse-detection wipes the user's chain on reuse
Weak secret	fail-closed boot check: `JWT_SECRET` ≥ 32 bytes (`openssl rand -base64 32`)
Brute-force on password	argon2id via `Bun.password` — memory-hard, OWASP-recommended

How the auth tokens behave over time



- **15 min — access token.** How often the cookie is silently swapped. You never see this; the frontend handles `401 → /auth/refresh → retry` transparently. F5 also implicitly triggers this via `App.svelte`'s `onMount → getMe()`.
- **7 days — refresh token.** How long you can sit idle before being bounced to the login form.

The 7-day clock slides forward on every successful refresh, so any authed activity once a week keeps the session alive indefinitely. Only **7 consecutive days of zero activity** ends the session — or one of: explicit sign-out (refresh row deleted), admin `DELETE FROM refresh_token WHERE user_id = ?` (compromise response), or reuse-detection firing.

Load-balancer config — `caddy/Caddyfile`

```
reverse_proxy backend-1:3000 backend-2:3000 {
  lb_policy round_robin
  health_uri /api/health
  health_interval 1s
  lb_try_duration 1s
  header_down +X-Upstream {http.reverse_proxy.upstream.hostport}
}
```

Active health checks (1 s interval) drop a dying backend out of rotation; `lb_try_duration` retries the survivor for in-flight requests caught mid-failure. The `X-Upstream` response header is what the demo scripts read to show which backend served each request.

Load-test comparison — `just load-test`

`oha -z 30s -c 50` against `GET /api/pastes`:

Metric	`main` (cookie sessions)	`jwt-auth` (this branch)
Throughput	3 443 req/s	11 663 req/s
p50 latency	14.7 ms	4.2 ms
p99 latency	41.5 ms	9.1 ms

`docker stats` mid-run shows the bottleneck shifting from Postgres (DB-bound, all containers ~115%) on `main` to Caddy (242% CPU, the proxy) on `jwt-auth`. Cutting one DB roundtrip per request gave ~3.4× throughput.