Project information

Created on
February 16, 2026

use indexes, increase uvicorn workers
Kevin Spath authored 2 hours ago

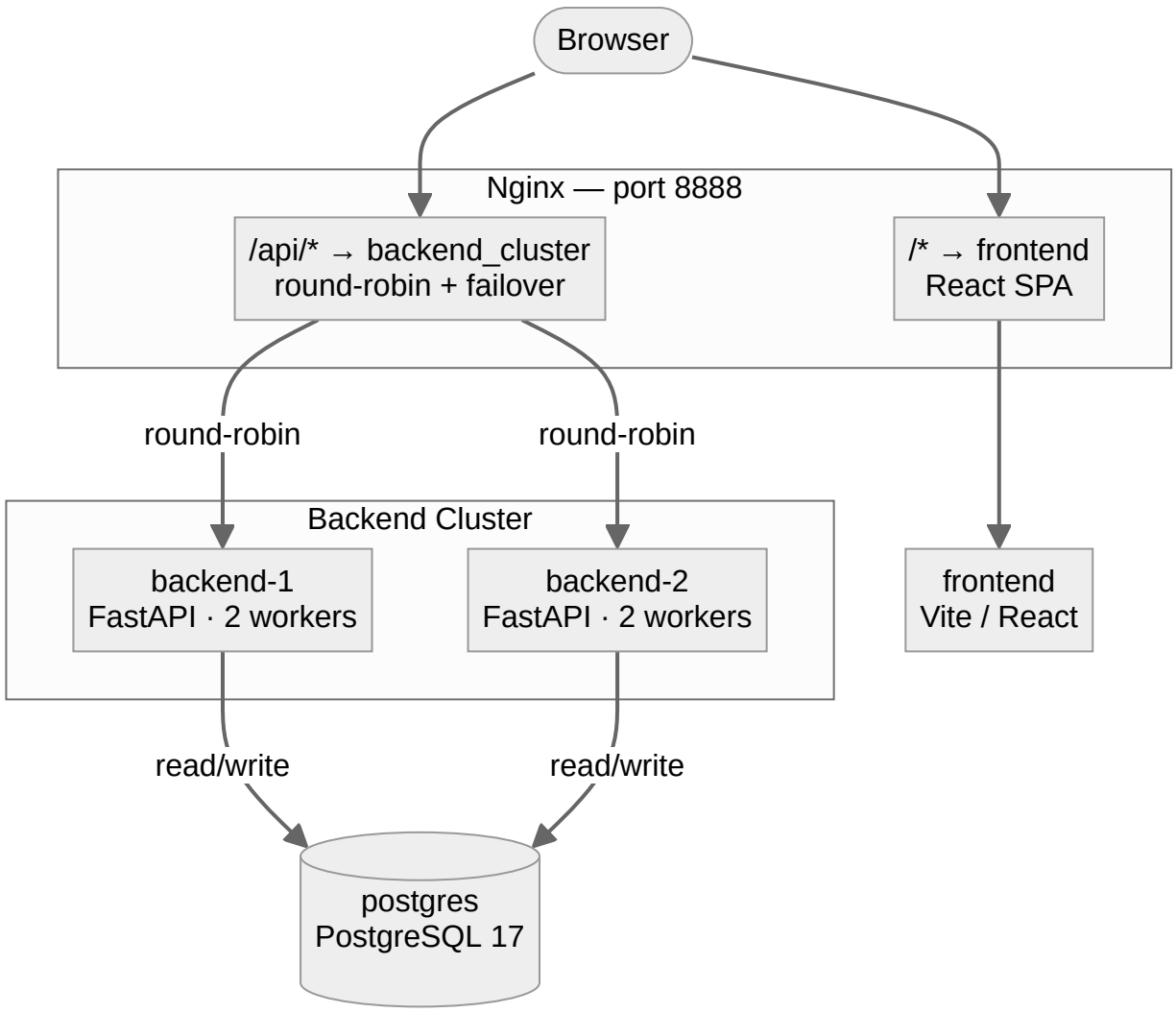
Name	Last commit	Last update
backend	use indexes, increase uvicorn workers	2 hours ago
frontend	fix	4 hours ago
loadtest	use indexes, increase uvicorn workers	2 hours ago
monitoring	working backend and frontend:transaction...	1 month ago
nginx	simplified	5 days ago
.env.example	fix	4 hours ago
.gitignore	fix	4 hours ago
README.md	use indexes, increase uvicorn workers	2 hours ago
docker-compose.yml	use indexes, increase uvicorn workers	2 hours ago
start.sh	fix	4 hours ago

README.md

Franz Finances

A brokerage platform built as a distributed systems challenge. Demonstrates load balancing, automatic backend failover, JWT authentication, and persistent storage — all brought up with a single command.

Architecture Overview



How Failover Works

Nginx resolves `backend-1` and `backend-2` via Docker's embedded DNS (`127.0.0.11`). The `resolver 127.0.0.11 valid=5s` directive re-queries DNS every 5 seconds so IP changes after a container restart are picked up automatically.

1. Both backends healthy → Nginx round-robins requests between them.
2. `backend-1` stops → Nginx gets a TCP error or timeout on the next request.
3. `proxy_next_upstream error timeout http_502 http_503 http_504` retries that request on `backend-2` immediately — the client never sees an error.
4. After `max_fails=3` in `fail_timeout=10s`, Nginx marks `backend-1` down and stops routing to it entirely.
5. When `backend-1` restarts, Nginx retries it after `fail_timeout` and returns it to rotation.

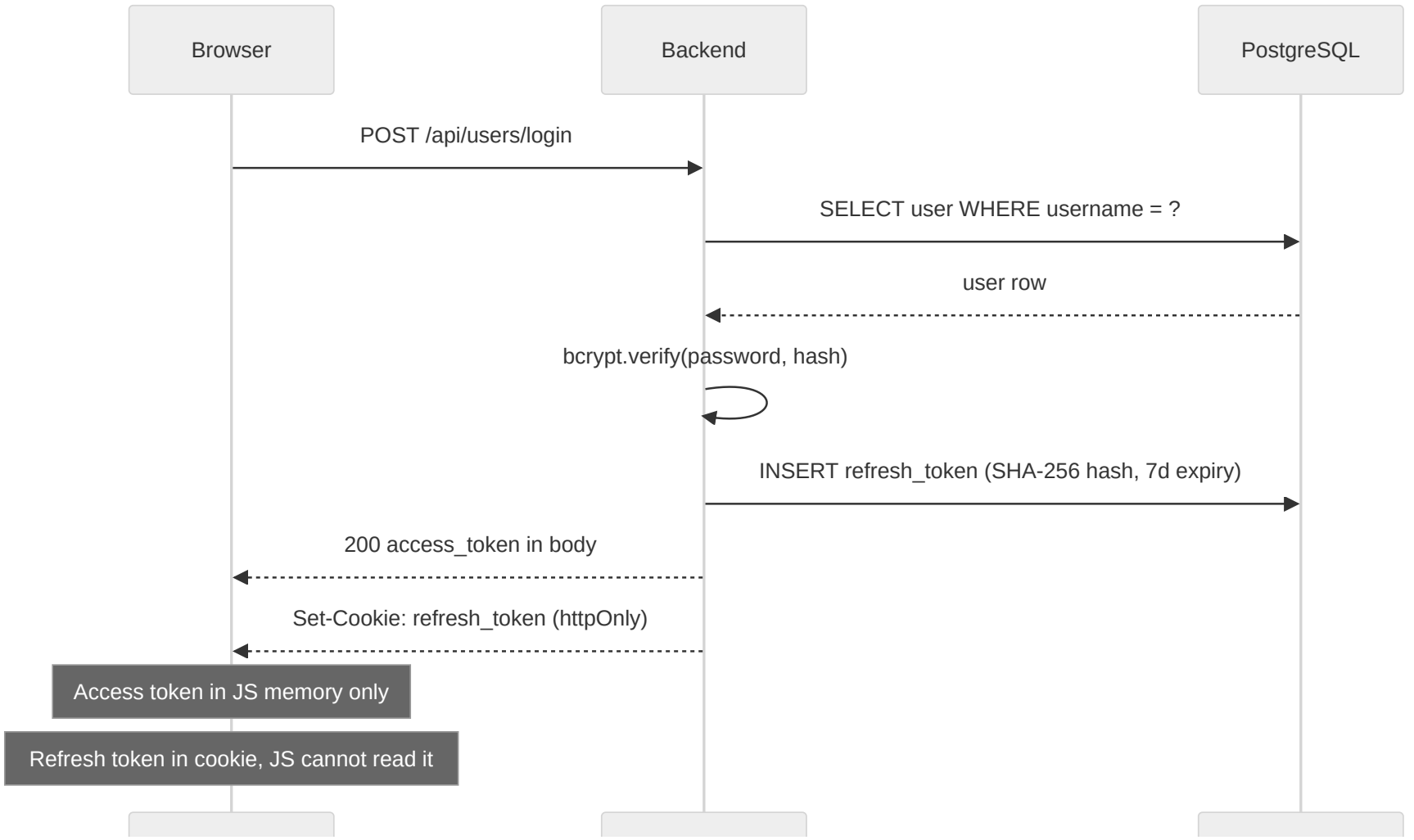
JWT Authentication

There are two token types in play at all times:

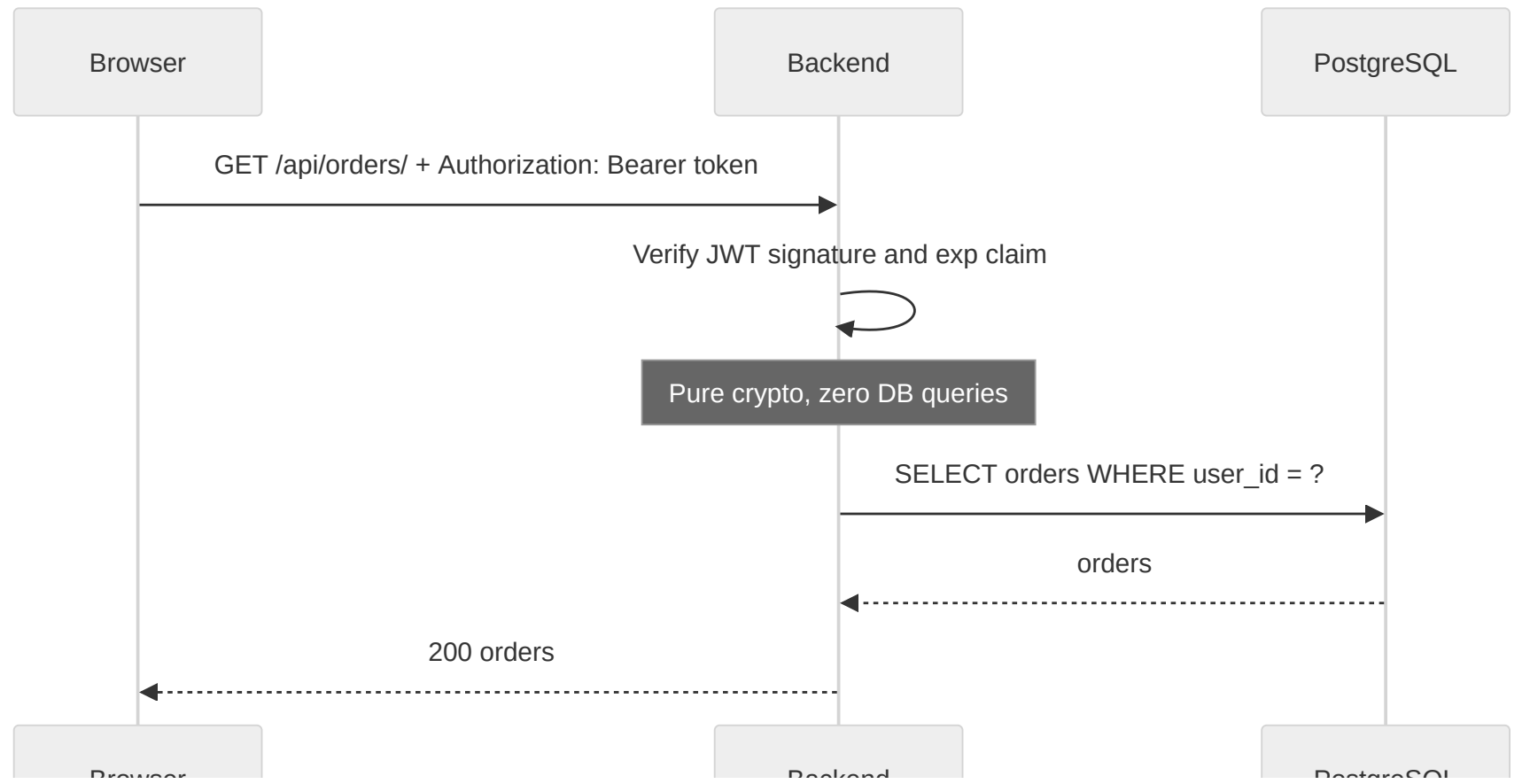
Token	Lifetime	Stored where	Purpose
Access token (JWT)	15 minutes	Browser memory only	Sent on every API request as `Authorization: Bearer …`
Refresh token	7 days	httpOnly cookie (invisible to JS)	Used only to get a new access token when the old one expires

---

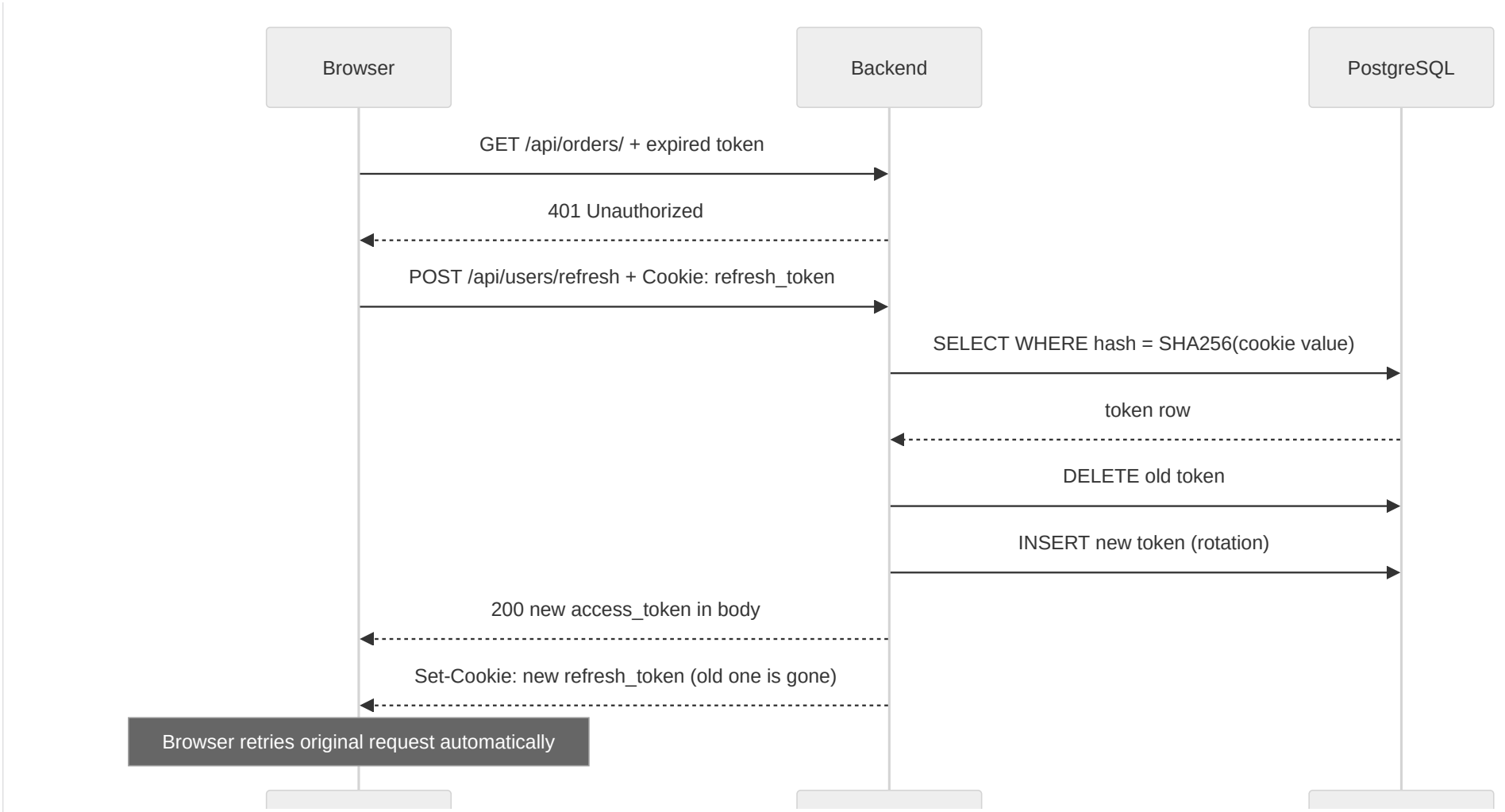
1 — Login



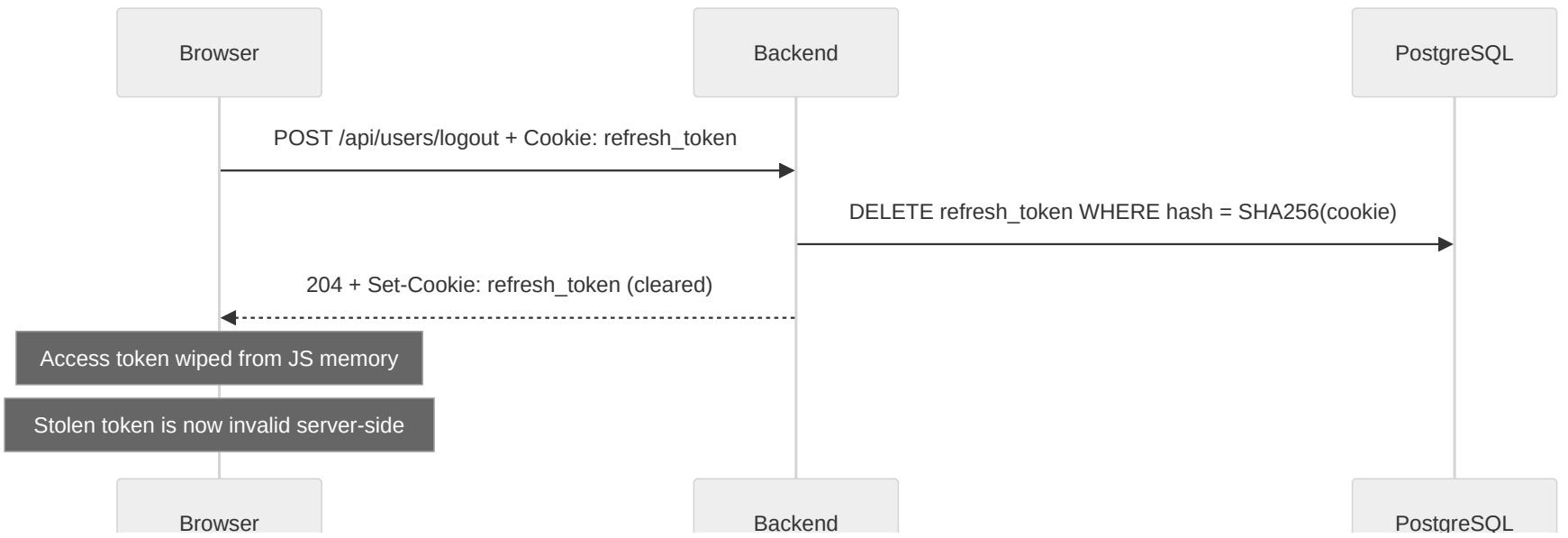
2 — Authenticated Request



3 — Token Refresh (access token expired)



4 — Logout



Why This Design (OWASP)

- **Access token in JS memory, not `localStorage`** — survives a page navigation but is gone on tab close. An XSS attack cannot steal it via `document.cookie` or `localStorage`.
- **Refresh token in httpOnly cookie** — JavaScript cannot read it at all. Only the browser sends it automatically to `/api/users`.
- **Refresh token stored hashed** — if the database is leaked, raw tokens are not exposed.
- **Rotation on every refresh** — each use of a refresh token invalidates it and issues a new one. A stolen token can only be used once before it becomes invalid.
- **Stateless access token validation** — the backend verifies the JWT signature cryptographically without a database query. This is what makes 1,100+ req/s possible.

Order Processing

- **Market orders** — executed synchronously when the order is placed: fetch live price from Yahoo Finance, deduct balance, update holdings, mark `COMPLETED`.
- **Limit orders** — saved as `PENDING`. Both backend instances poll every 30 seconds for triggered limit orders. `SELECT FOR UPDATE SKIP LOCKED` ensures two backends never process the same order concurrently.

Component Versions

Component	Image	Version
Backend (FastAPI)	`python:3.13-slim`	Python 3.13

Component	Image	Version
Frontend (React/Vite)	`node:24-slim`	Node 24 LTS
Database	`postgres:18`	PostgreSQL 18
Load Balancer	`nginx:1.30`	Nginx 1.30

Quick Start

Prerequisites: Docker Engine 24+ and Docker Compose v2

```
git clone <repository-url>
cd franz-finances
docker compose up --build
```

All services start in dependency order. Ready when both backends log `Application startup complete.`

Access points

Service	URL
Frontend App	http://localhost:8888
Nginx health	http://localhost:8080/nginx_health
PostgreSQL	`localhost:5432` — `franz / franz-secret`

Load Test

Run the three-phase load test (requires `wrk`):

```
./loadtest/run.sh
```

Phases:

1. **Baseline** — both backends running, measures peak throughput
2. **Failover** — kills `backend-1` after 5s while `wrk` keeps running; zero errors expected
3. **Recovery** — `backend-1` restarted, back to full capacity

Latest results (8 threads, 50 connections, 30s per phase):

Phase	req/s	avg latency	p99 latency
Baseline (both backends)	~1,980	27ms	107ms
Failover (backend-1 killed at 5s)	~1,900	—	—
Recovery (both backends again)	~1,980	—	—

Bottleneck: PostgreSQL — JWT validation is stateless (pure crypto, no DB), but `GET /api/orders/` still needs one DB query to fetch the user's orders. The single PostgreSQL instance is the throughput ceiling.

Indexes added for production-scale queries:

- `orders(user_id, created_at)` — composite index covering `WHERE user_id = ? ORDER BY created_at DESC`
- `orders(status, limit_price)` — covering the limit order poll: `WHERE status = PENDING AND limit_price IS NOT NULL`
- `holdings(user_id)` — covering `WHERE user_id = ?` on portfolio fetch

Baseline only (no container kill):

```
./loadtest/run.sh --baseline
```

Testing Failover Manually

```
# Terminal 1 — watch which backend handles each request
while true; do
  curl -s http://localhost:8888/api/health | python3 -m json.tool
  sleep 1
done

# Terminal 2 — kill one backend
docker compose stop backend-1
# Terminal 1 continues without errors — all traffic moves to backend-2

# Bring it back
docker compose start backend-1
# Round-robin resumes automatically
```

Development

```
# Rebuild after code changes
docker compose up --build

# Rebuild a single service
docker compose up --build backend-1 backend-2

# Database access
docker compose exec postgres psql -U franz -d franzfinances

# Useful queries
SELECT * FROM orders ORDER BY created_at DESC LIMIT 20;
SELECT * FROM holdings;

# Follow logs
docker compose logs -f backend-1 backend-2
docker compose logs -f nginx
```

Reset Everything

```
# Remove all containers and volumes (⚠ deletes all data)
docker compose down -v
docker compose up --build
```

Project Structure

```
franz-finances/
├── backend/                  # FastAPI application
│   ├── app/
│   │   ├── main.py           # app factory, lifespan, limit-order background task
│   │   ├── config.py         # settings (DATABASE_URL, JWT_*, INSTANCE_ID)
│   │   ├── auth.py           # JWT encode/decode, stateless get_current_user
│   │   ├── models.py         # SQLModel ORM models
│   │   ├── schemas.py        # Pydantic request/response schemas
│   │   ├── processor.py      # order execution + limit-order polling
│   │   ├── resolver.py       # ISIN / ticker symbol lookup
│   │   ├── database.py       # async SQLAlchemy engine (pool_size=20)
│   │   └── routers/
│   │       ├── orders.py     # order placement, preview, history
│   │       ├── portfolio.py  # holdings and balance
│   │       └── users.py      # register, login, refresh, logout
│   ├── Dockerfile
│   ├── entrypoint.sh         # sets INSTANCE_ID, starts uvicorn --workers 2
│   └── requirements.txt
├── frontend/                 # React + Vite + Tailwind
```

```
│   ├── src/
│   │   ├── App.jsx
│   │   ├── api.js
│   │   └── ...
│   └── Dockerfile
├── nginx/
│   └── nginx.conf            # upstream, DNS resolver, failover config
├── loadtest/
│   └── run.sh                # wrk-based 3-phase load test
└── docker-compose.yml        # full-stack orchestration
```

API Reference

Authentication

Method	Path	Description
POST	`/api/users/register`	Create account
POST	`/api/users/login`	Login, returns JWT access token + refresh cookie
POST	`/api/users/refresh`	Rotate refresh token, return new access token
POST	`/api/users/logout`	Revoke refresh token

Trading

Method	Path	Description
POST	`/api/orders/preview`	Preview total cost before placing
POST	`/api/orders/`	Place market or limit order
GET	`/api/orders/`	List own orders
GET	`/api/orders/{id}`	Get single order
PATCH	`/api/orders/{id}`	Modify pending limit order price
DELETE	`/api/orders/{id}`	Cancel pending limit order
GET	`/api/orders/search?q=AAPL`	Symbol / ISIN search
GET	`/api/orders/price/{ticker}`	Live market price

Portfolio

Method	Path	Description
GET	`/api/portfolio/`	Balance and current holdings

System

Method	Path	Description
GET	`/api/health`	Returns `{"status":"ok","instance":"backend-1"}`

Fault Tolerance Summary

Component	Redundancy	Tolerated failures
Backend	2 instances (2 uvicorn workers each), Nginx round-robin + failover	1 backend
Frontend	1 instance, served via Nginx	—

Component	Redundancy	Tolerated failures
Database	Single PostgreSQL instance (`max_connections=200`)	—