Project information

Created on
February 17, 2026

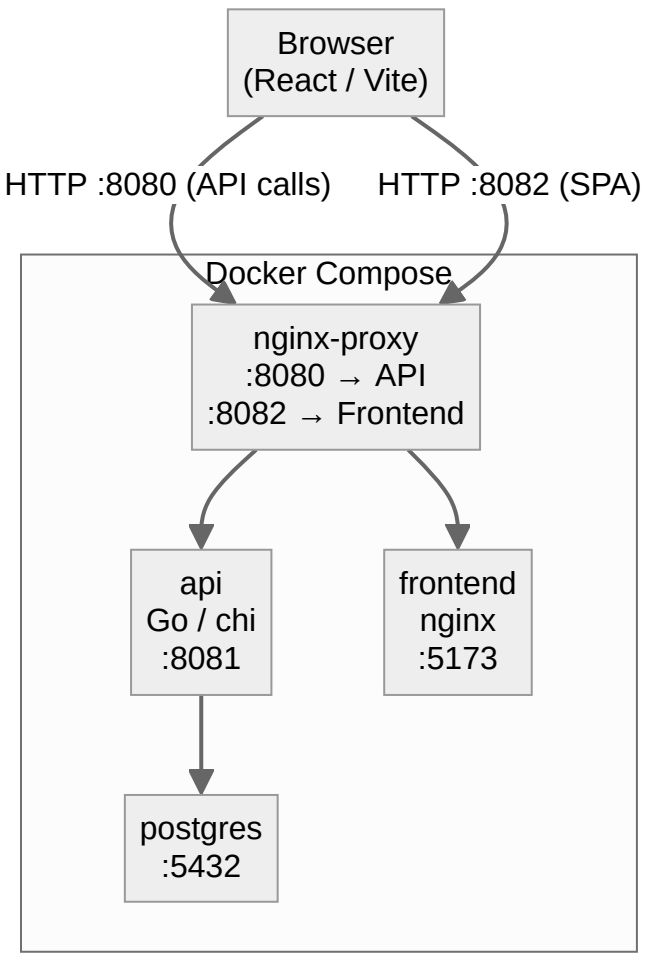
Name	Last update
api	4 hours ago
frontend	1 hour ago
loadbalancer	1 hour ago
postgres	4 hours ago
.gitigonre	4 hours ago
Makefile	4 hours ago
README.md	just now
compose.yml	1 hour ago

README.md

2Password

A self-hosted password manager with client-side encryption. Passwords are encrypted in the browser before being sent to the server — the plaintext master password and decrypted credentials never leave the client.

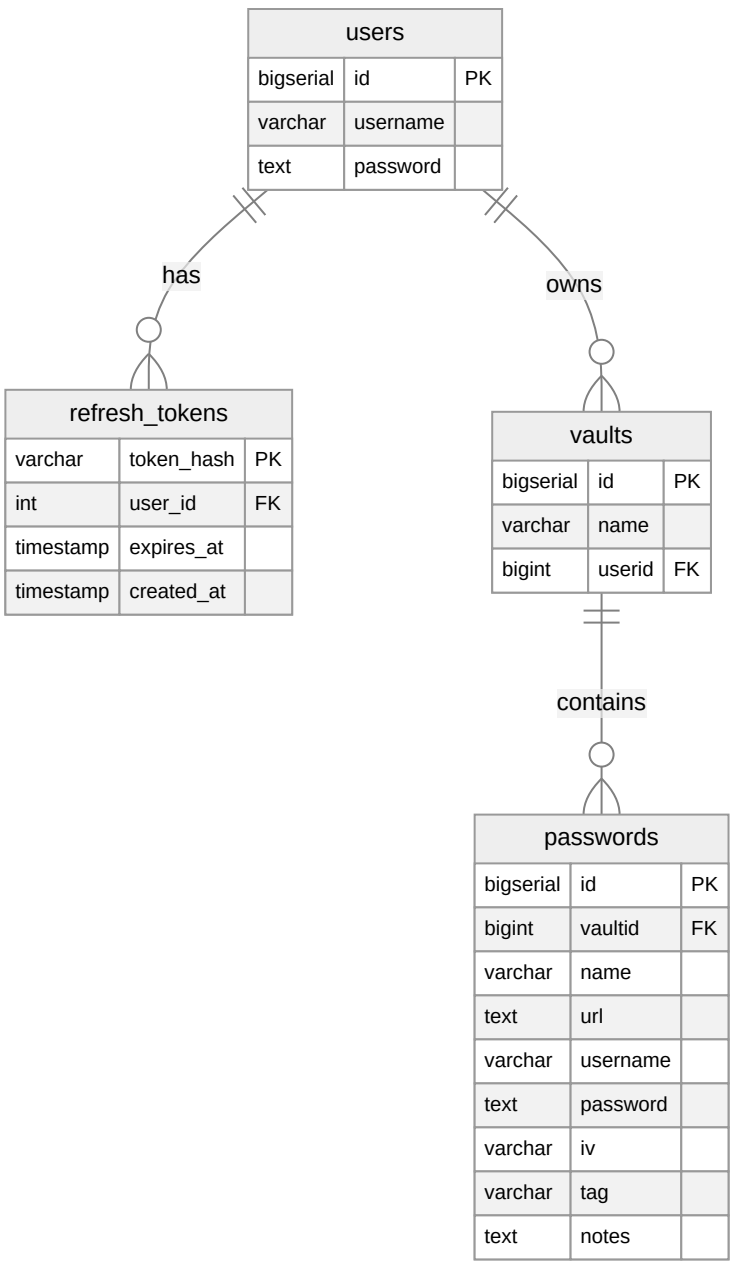
Architecture



Services

Service	Image	Port	Role
`nginx-proxy`	nginx	8080, 8082	Reverse proxy / load balancer
`api`	debian:oldstable-slim	8081 (internal)	Go REST API
`frontend`	nginx	5173 (internal)	Serves pre-built Vite SPA
`postgres`	postgres:latest	5432	Persistent storage

Data model



Security model

Client-side encryption

All password data is encrypted in the browser before being transmitted or stored. The server never sees plaintext credentials.

- **Key derivation**: PBKDF2-HMAC-SHA256, 250 000 iterations (OWASP 2023 minimum), salted with `passwordmanager:<userId>`
- **Encryption**: AES-GCM 256-bit; a random 12-byte IV is generated per entry
- **Master key**: held only in a JS `Map` in memory, never serialised or sent to the server

Server-side security

- **User passwords**: bcrypt (default cost) stored in the database
- **Authentication**: short-lived JWT access tokens (15 min, HS256) + long-lived refresh tokens (7 days)
- **Refresh token rotation**: each `/refresh` call invalidates the old token and issues a new one; tokens are stored in the DB as SHA-256 hashes

API reference

All routes except `/signup`, `/login`, `/refresh`, and `/info` require a `Bearer` JWT in the `Authorization` header.

```
POST   /signup
POST   /login
POST   /refresh
GET    /info
```

```
GET    /users/
GET    /users/{userID}
DELETE /users/{userID}

GET    /users/{userID}/vaults/
POST   /users/{userID}/vaults/
GET    /users/{userID}/vaults/{vaultID}
DELETE /users/{userID}/vaults/{vaultID}

GET    /users/{userID}/vaults/{vaultID}/passwords/
POST   /users/{userID}/vaults/{vaultID}/passwords/
GET    /users/{userID}/vaults/{vaultID}/passwords/{passwordID}
PUT    /users/{userID}/vaults/{vaultID}/passwords/{passwordID}
DELETE /users/{userID}/vaults/{vaultID}/passwords/{passwordID}
```

Running locally

Prerequisites: Docker + Docker Compose, Go (for building the API), Node.js (for building the frontend).

```
# 1. Build the Go API binary
cd api
go build -o api .
cd ..

# 2. Build the frontend
cd frontend
npm install
npm run build
cd ..

# 3. Start the stack
docker compose up
```

The app is then available at:

- Frontend: http://localhost:8082
- API: http://localhost:8080

Database initialisation

On first run, apply the schema:

```
docker exec -i 2password-db psql -U myuser -d mydatabase < postgres/db-init.sql
```

Project structure

```
.
├── api/                    # Go backend
│   ├── main.go
│   ├── http/http.go        # Route definitions and handlers
│   ├── db/db.go            # PostgreSQL queries (pgx/v5)
│   ├── crypto/crypto.go    # AES-GCM + scrypt utilities (server-side)
│   └── structs/            # Shared types
├── frontend/               # React + Vite SPA
│   ├── src/
│   │   ├── crypto.js       # Client-side PBKDF2 + AES-GCM (Web Crypto API)
│   │   ├── masterKey.js    # In-memory key cache
│   │   ├── api.js          # Fetch wrappers + token refresh logic
│   │   └── components/     # Login, Signup, Dashboard, VaultList, PasswordList, …
│   └── nginx.conf
├── loadbalancer/
│   └── nginx.conf          # Reverse proxy config (ports 8080 / 8082)
├── postgres/
│   └── db-init.sql         # Schema (users, vaults, passwords, refresh_tokens)
```

```
└── compose.yml
```