**Project information**

**Created on**
March 01, 2026

| Name | Last update |
|---|---|
| backend | 24 minutes ago |
| frontend | 24 minutes ago |
| nginx | 1 month ago |
| .env.example | 2 hours ago |
| .gitignore | 1 month ago |
| README.md | 24 minutes ago |
| docker-compose.yml | 2 hours ago |
| load-test-demo.sh | 3 hours ago |
| pnpm-lock.yaml | 16 hours ago |

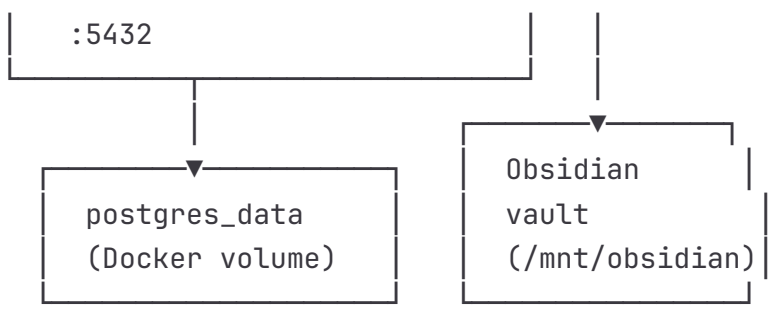
**README.md**

# Stash

Stash is a distributed "brain-buffer" system designed for high-efficiency note-taking and task offloading. The idea is to be able to "stash" fleeting ideas as fast as possible — then get back to what you were doing.

Notes are parsed from raw text into structured data with priority levels, tags, and categories. Stashed notes can be filtered, released, or exported directly into an Obsidian vault as markdown files with YAML frontmatter.

## Architecture




## Tech Stack

- **Frontend:** React, TypeScript, Vite
- **Backend:** Node.js, Fastify, Prisma ORM
- **Database:** PostgreSQL 18 (Alpine)
- **Auth:** Access/refresh token pair — short-lived access tokens (15 min, Bearer header) + long-lived refresh tokens (7 days, httpOnly cookie). Passwords hashed with argon2.
- **Reverse proxy:** Nginx (round-robin load balancing)
- **Obsidian integration:** direct filesystem export via bind-mounted vault
- **Containerisation:** Docker Compose with YAML anchors for shared config

## Services

The system runs as six Docker Compose services with an orchestrated boot order:

| # | Service | Role | Details |
|---|---|---|---|
| 1 | `stash-db` | Database | PostgreSQL 18 on Alpine. Includes a `pg_isready` health check that downstream services wait on. Data persists via a named Docker volume. |
| 2 | `migrator` | Schema setup | One-shot container that runs `prisma migrate deploy` and `prisma db seed`, then exits. Guarantees the database schema and seed user exist before any API instance starts. |
| 3 | `one-api` | API instance | Fastify server with Prisma ORM and JWT auth. Stateless — connects to the shared database. Has the Obsidian vault bind-mounted at `/mnt/obsidian`. |
| 3 | `two-api` | API instance | Identical to `one-api`. Both instances run the same code; only the `HOSTNAME` env var differs (for log identification). Shares the same Obsidian vault mount. |
| 4 | `load-balancer` | Reverse proxy | Nginx on Alpine. Distributes traffic across both API instances via round-robin. The only entry point for API requests. |
| 5 | `frontend` | Web UI | Vite dev server with hot reload via bind mount. Connects to the API through the load balancer. |

Shared environment variables (database URL, JWT secret, connection pool settings) are defined once via a YAML anchor (`x-api-env`) and merged into both API instances.

## API Endpoints

| Method | Path | Auth | Description |
|---|---|---|---|
| `POST` | `/api/login` | No | Authenticate with email/password. Returns a short-lived access token in the response body and sets an httpOnly refresh token cookie. |
| `POST` | `/api/refresh` | No | Exchange a valid refresh token cookie for a new access token (and a rotated refresh cookie). Used for silent session restoration on page load. |
| `GET` | `/api/status` | No | Health check. Returns `{ status, message, node }` where `node` is the `HOSTNAME` env var (useful for confirming load balancer routing). |
| `GET` | `/api/notes` | Yes | Fetch notes. Optional query filters: `priority` (`NORMAL`, `IMPORTANT`, `URGENT`), `tag`, `category`, `released` (`true`/`false`). Results are ordered by priority (desc) then creation date (desc). |
| `POST` | `/api/notes` | Yes | Create a new note from raw text (`{ "raw": "..." }`). The raw input is parsed into structured data with content, priority, tags, and category. |
| `POST` | `/api/notes/:id/release` | Yes | Release a stashed note (sets `releasedAt` timestamp). |

| Method | Path | Auth | Description |
|---|---|---|---|
| `POST` | `/api/notes/:id/obsidian` | Yes | Export a note to the Obsidian vault as a markdown file with YAML frontmatter (priority, category, tags, created date). Returns 501 if the vault is not mounted. |

## Auth Flow

Login returns a short-lived access token (15 min) in the response body alongside an httpOnly `refreshToken` cookie (7 days). The frontend stores the access token in `localStorage` and attaches it as a `Bearer` header on every API request.

On app load, the frontend silently calls `POST /api/refresh` to restore a session from the cookie — no login screen is shown if the cookie is still valid. If any request returns 401, the client automatically retries the refresh once before redirecting to `/login`. Concurrent 401s share a single in-flight refresh request to avoid race conditions.

Passwords are hashed with argon2. The same `JWT_SECRET` is used across all API instances so tokens are valid regardless of which instance the load balancer routes to.

## Date Scheduling & Calendar Export

A date embedded anywhere in a note's raw text is recognised by the parser and stored as `scheduledAt` / `allDay` on the note. The date token is stripped from the displayed content — so `call dentist 1506` shows as "call dentist" with a date chip of `15.06`.

**Supported formats** (all standalone, i.e. surrounded by whitespace or at string boundaries):

| Format | Example | Notes |
|---|---|---|
| `DD.MM.YYYY` | `15.06.2026` | Full European date |
| `DD.MM.YYYY hh:mm` | `15.06.2026 14:30` | Full European date + time |
| `YYYY-MM-DD` | `2026-06-15` | ISO date |
| `YYYY-MM-DD hh:mm` | `2026-06-15 14:30` | ISO date + time |
| `YYYYMMDD` | `20260615` | Compact 8-digit |
| `DDMMYYYY` | `15062026` | Compact European |
| `DDMM hhmm` | `1506 1430` | Short date + time (current year) |
| `DD.MM` | `15.06` | Short European (current year) |
| `DDMM` | `1506` | Plain short (current year) |

Notes with a recognised date show a date chip in the card footer. A **calendar** button appears alongside it — clicking it generates and downloads an `.ics` file directly in the browser (no server call). The event title is the note content with the date token removed. All-day events get a 24-hour duration; timed events default to 1 hour.

## Obsidian Sync

Stash can export individual notes to a local Obsidian vault. The `OBSIDIAN_VAULT` environment variable points to your vault's location on the host machine, and Docker bind-mounts it into both API containers at `/mnt/obsidian`.

When you hit `POST /api/notes/:id/obsidian`, the API:

1. Checks that the vault mount is accessible (returns 501 if not).
2. Fetches the note with its tags and category.
3. Generates a markdown file with YAML frontmatter containing the note's metadata.
4. Writes it to the vault directory.

The exported file looks like:

```
---
priority: IMPORTANT
category: project-ideas
tags: [backend, rust]
created: 2026-05-10T14:32:00.000Z
---
```

```
Rewrite the parser in Rust for fun
```

The filename is derived from the first 40 characters of the note content. Since the vault is bind-mounted from the host filesystem, Obsidian picks up new files automatically via its built-in file watcher.

## Getting Started

### Prerequisites

- Docker and Docker Compose
- Node.js (for local development outside Docker)

### Setup

1. Clone the repository:

```
git clone <repo-url>
cd stash
```

2. Create a `.env` file from the example:

```
cp .env.example .env
```

3. Fill in your `.env` values:

```
JWT_SECRET="your-secret-key"
SEED_EMAIL="you@example.com"
SEED_PASSWORD="your-password"
DB_USER=stash_admin
DB_PASSWORD=super_secret_password
DB_NAME=stash_db
DB_POOL_LIMIT=20
DB_POOL_TIMEOUT=15
OBSIDIAN_VAULT="/Users/yourname/Library/Mobile Documents/iCloud~md~obsidian/Documents/obsidian"
```

| Variable | Description |
|---|---|
| `JWT_SECRET` | Secret key for signing/verifying JWT tokens. Must be the same across all API instances. |
| `SEED_EMAIL` / `SEED_PASSWORD` | Credentials for the initial user created by the migrator. |
| `DB_USER` / `DB_PASSWORD` / `DB_NAME` | PostgreSQL credentials and database name. |
| `DB_POOL_LIMIT` | Max database connections per API instance (Prisma connection pool). |
| `DB_POOL_TIMEOUT` | Seconds Prisma waits for a free connection before throwing an error. |
| `OBSIDIAN_VAULT` | Absolute path to your local Obsidian vault. The `/stash` subdirectory is mounted into containers. |

4. Start the system:

```
docker compose up -d
```

5. Open `http://localhost:5173` in your browser.

The migrator service automatically handles database setup and seeding on first boot.

## Load Testing

A load testing script is included for benchmarking with hey:

```
# Install prerequisites
brew install hey jq
```

```
# Run the demo (params: total requests, concurrency)
chmod +x load-test-demo.sh
./load-test-demo.sh 1000 50
```

The script reads credentials from `.env` automatically, runs GET, filtered GET, and POST benchmarks at the specified load, and includes a fault tolerance demo that kills an API instance mid-test to demonstrate the load balancer routing around failures.