Traumabomen — Design Document

Date: 2025-02-09 **Status:** Draft **Author:** Brainstorming session

1. Overview

Traumabomen is a private, zero-knowledge encrypted web application where users map intergenerational trauma onto a visual family tree. The primary purpose is personal reflection — a journaling/mapping tool for self-insight into family patterns.

Users build a family tree, attach trauma events to persons, and use a timeline overlay to spot recurring patterns across generations. All sensitive data is encrypted client-side; the server never sees plaintext.

2. Domain Model

Person

- `id` : UUID
- `name` : string
- `birth_year` : integer (approximate)
- `death_year` : integer (optional)
- `gender` : string
- `is_adopted` : boolean
- `notes` : string (optional)

Relationship (edge between two Persons)

Typed and directional. Types:

- **Biological parent** — birth parent connection
- **Step-parent** — partner of a biological parent, no biological link
- **Adoptive parent** — legally/emotionally parenting, non-biological

- **Biological sibling** — shared both parents
- **Step-sibling** — connected through step-parent relationship
- **Partner** — romantic/marital relationship (see temporal model below)

Half-sibling relationships are inferred: two persons sharing exactly one biological parent. Not stored as a separate edge type.

Parent-type relationships carry an optional `active_period` (start/end year) since step-parent relationships begin and end with the partner relationship.

Temporal Partner Relationships

Partner edges contain a list of `RelationshipPeriod` entries:

- `start_year` : integer (approximate)
- `end_year` : integer (optional, null = ongoing)
- `status` : `together` | `married` | `separated` | `divorced`

This supports real-world complexity: a couple that married, divorced, and later reunited gets multiple periods on the same edge. Children born during different periods are contextually placed.

TraumaEvent

- `id` : UUID
- `person_ids` : list of UUIDs (attached to one or more Persons)
- `title` : string
- `description` : string (free text)
- `category` : string (e.g., loss, abuse, addiction, war, displacement, illness, poverty)
- `approximate_date` : string (year or period)
- `severity` : integer (user-defined scale)
- `tags` : list of strings (optional)

Pattern (deferred from MVP)

- `id` : UUID
- `name` : string

- `description` : string
- `linked_event_ids` : list of TraumaEvent UUIDs

An annotation layer linking multiple TraumaEvents across generations to mark recurring themes.

3. Architecture

Stack

Layer	Technology
Frontend	Vite + React + React Router + TypeScript
Tree visualization	React Flow + Dagre layout
Timeline visualization	D3.js
i18n	react-i18next
Backend	FastAPI + SQLAlchemy + PostgreSQL
Auth	JWT (access + refresh tokens)
Client-side encryption	Web Crypto API (AES-256-GCM) + Argon2id (WASM)
State management	TanStack Query (React Query)

Deployment

- Frontend: static assets to S3/CDN/Vercel static
- Backend: fly.io, Railway, or VPS

Zero-Knowledge Encryption Flow

1. **Registration:** User provides email/password (auth) and a separate encryption passphrase. Client generates a salt (stored server-side). Passphrase + salt derive an AES-256 key via Argon2id. Key is held in memory only.
2. **Login:** Authenticate via JWT → fetch salt → prompt for passphrase → derive key → decrypt tree data.
3. **Data operations:** All sensitive fields encrypted client-side before any API call. Server

stores opaque ciphertext blobs.

4. **Tab close / logout:** Key is garbage collected. No persistence.
5. **Passphrase change:** Decrypt all blobs with old key → re-encrypt with new key → bulk sync.
6. **Passphrase lost = data lost.** UI must make this explicit during registration with a confirmation step.

Encryption Module (`/lib/crypto.ts`)

- `deriveKey(passphrase, salt)` → AES-256-GCM key via Argon2id (`argon2-browser` WASM)
- `encrypt(plaintext, key)` → random IV + AES-256-GCM → `{ iv, ciphertext }` as base64
- `decrypt(encryptedBlob, key)` → extract IV, decrypt, return plaintext
- `generateSalt()` → used during registration

4. API Design

The backend is a thin encrypted document store with auth. No domain logic server-side (content is opaque).

Endpoints

Auth:

- `POST /auth/register`
- `POST /auth/login`
- `POST /auth/refresh`

Resources (all payloads: `{ id, encrypted_data, metadata }`):

- `GET/POST/PUT/DELETE /trees`
- `GET/POST/PUT/DELETE /trees/{id}/persons`
- `GET/POST/PUT/DELETE /trees/{id}/relationships`
- `GET/POST/PUT/DELETE /trees/{id}/events`
- `GET/POST/PUT/DELETE /trees/{id}/patterns`

Bulk sync:

- `POST /trees/{id}/sync` — batch of creates, updates, deletes across all entity types in a single transaction

Server validates: auth, ownership, structural integrity (referenced UUIDs exist). Server cannot validate content (encrypted).

5. Frontend Architecture

Routing

- `/login` , `/register` — auth flows
- `/trees` — tree list (MVP: single tree)
- `/trees/{id}` — main workspace, tree view
- `/trees/{id}/timeline` — timeline view

Key Components

- **`<EncryptionProvider>`** — React context holding derived key in memory. Exposes `encrypt()` / `decrypt()` . Wraps authenticated app.
- **`<TreeCanvas>`** — React Flow instance. Person nodes, relationship edges. Dagre auto-layout. Drag-to-create relationships, zoom, pan.
- **`<PersonNode>`** — Custom React Flow node. Name, years, adoption icon, trauma event badges (color-coded by category).
- **`<PersonDetailPanel>`** — Slide-out panel. Edit person fields, trauma events, relationship periods. Encrypt-then-save.
- **`<TimelineView>`** — D3 horizontal timeline. Generational bands as rows, trauma events as markers, partner periods as horizontal bars.
- **`<PatternEditor>`** — Deferred from MVP.

Relationship Visual Styles

- Solid lines: biological relationships
- Dashed lines: step/adoptive relationships
- Small icons/labels on edges for type clarity

6. Internationalization

- `react-i18next` with JSON translation files
- `/locales/en/translation.json` — English (base, all keys)
- `/locales/nl/translation.json` — Dutch
- Language detection: browser preference, overridable in settings
- All UI strings via `t('key')` from day one
- Flat namespaced keys: `tree.addPerson`, `trauma.category.addiction`, `relationship.type.stepParent`
- Date formatting via `Intl.DateTimeFormat` respecting locale

7. Testing Strategy

Unit Tests (Vitest)

- Encryption module: round-trip encrypt/decrypt, key derivation determinism, IV uniqueness
- Domain logic: half-sibling inference, relationship period validation (no overlapping periods), pattern linking

Component Tests (React Testing Library)

- PersonNode rendering (adopted, trauma badges, various states)
- PersonDetailPanel CRUD flows
- Relationship period editor (add/remove periods)
- Tested against decrypted in-memory state, no crypto in component tests

Integration Tests (Playwright)

- Full journeys: register → passphrase → create tree → add persons → relationships → trauma events → timeline view
- Crypto round-trip: logout → login → passphrase → verify decryption
- Failure path: wrong passphrase → graceful error

Backend Tests (pytest)

- API endpoints: encrypted blobs stored/returned untouched
- Auth flows
- Bulk sync transactionality (partial failure → rollback)
- Ownership isolation (user A cannot access user B's trees)

8. MVP Scope

In MVP

- Auth (email/password + encryption passphrase)
- Single tree per user
- Person CRUD with all relationship types
- Temporal partner relationships
- Trauma event CRUD with categories
- Tree view (React Flow + Dagre layout)
- Timeline view (basic D3)
- Zero-knowledge encryption
- English + Dutch
- Bulk sync endpoint

Deferred

- Multiple trees per user
- Pattern editor
- OAuth/social login
- GEDCOM import
- PDF/image export
- Custom category management
- Collaborative/shared trees
- Passphrase recovery hints

- Offline-first with service worker
- Additional languages