ChatGPT

<!-- This file provides a comprehensive implementation-ready documentation set for **PROJECT_NAME**, a web application that allows users to visually build and download genealogical trees. Every section has been crafted with rationale and supporting references from authoritative sources. Citations from third-party sources are provided using tether IDs. -->

PROJECT_NAME – Implementation-Ready Documentation

Executive Summary

PROJECT_NAME is a browser-based application that enables anyone to create, edit and visualize a family tree without needing to sign up. The product empowers casual family enthusiasts and serious genealogists alike to map complex relationships (biological, adoptive, step, same-sex partnerships, etc.) and export their work for safe keeping or further research. The MVP focuses on a **local-first** experience: users build trees offline using an intuitive drag-and-drop canvas backed by IndexedDB, then export their data as a JSON file. An optional cloud sync mode (post-MVP) adds collaboration and backup through a lightweight API. Throughout the design the application prioritizes **privacy**, **accessibility** (WCAG 2.2 AA), **internationalization** (English/Spanish/Portuguese), and **performance** (supporting trees with thousands of nodes). Supporting documentation below includes product requirements, UX flows and wireframes, data models, export/import formats, architecture diagrams, API stubs, component specifications, validation rules, testing plan, DevOps guidance, a roadmap, appendices, and a final self-check.

Table of Contents

Product Requirements Document (PRD)

Problem Statement and Goals

Genealogy enthusiasts often rely on complex desktop software or web services that require accounts, impose strict data models, or lock data behind proprietary formats. Casual users may only wish to map a small family tree for a school project, while serious family historians need to model complicated relationships such as adoptions, blended families, or same-sex marriages. They need an easy-to-use tool that works offline, respects privacy, and allows exporting their tree for use elsewhere. **PROJECT_NAME** aims to fill this gap by providing a local-first, privacy-respecting, intuitive web application for building genealogical trees.

Personas & Jobs-to-Be-Done

Persona	Description	Jobs-to-Be-Done
Casual User	A non-technical person building a small tree to learn about their family history.	Create a new project, add parents/children/siblings, mark unknown information, export a file to share with relatives.
Family Historian	An enthusiast building an extensive lineage including step-families and complex relationships.	Model multiple marriages, adoptions and same-sex partnerships; merge duplicate entries; attach notes and sources; export to GEDCOM for use in other tools.
Researcher	A genealogist or academic tracing historical data sets.	Enter approximate dates (about, before, after) [1], track sources, export CSV for analysis, and import existing JSON/CSV files.

User Stories and Acceptance Criteria

The following representative user stories illustrate the MVP scope. Acceptance criteria are written in Gherkin-style for clarity.

1. **Create a new family tree**
    - *As a* user
    - *I want* to start a new project without signing up
    - *So that* I can immediately begin adding family members
    - **Acceptance:**

```
Given I am on the home page
When I click “Create New Tree”
Then a new empty canvas opens with a default root person card ready
for editing
```

2. **Add a person and define a relationship**

    - *As a* user
    - *I want* to add a new person and link them as a parent, child, spouse/partner, or sibling
    - *So that* the family structure reflects real relationships
    - **Acceptance:**

```
Given a person is selected on the canvas
When I choose “Add Parent” from the context menu
And I enter the new person’s details
Then a new person node appears connected above the selected person
And the relationship type is marked (biological, adoptive, step,
partner)
```

3. **Enter approximate dates**

    - *As a* researcher
    - *I want* to enter “about 1900” or “between 1870 and 1880” for dates
    - *So that* uncertain historical events can be recorded
    - **Acceptance:**

```
Given I am editing a birth event
When I select “Approximate Date” and choose “About” with the year
1900
Then the event displays as “About 1900” and the export file includes
type=“about” and value=“1900”
```

4. **Undo and redo edits**

    - *As a* user
    - *I want* to undo or redo my recent changes
    - *So that* mistakes can be corrected easily
    - **Acceptance:**

```
Given I added a person
When I press Ctrl+Z
Then the person is removed from the canvas
And when I press Ctrl+Shift+Z
Then the person returns
```

5. **Export data**

    - *As a* user
    - *I want* to download my family tree as a JSON file
    - *So that* I can back it up or import it into other software

- **Acceptance:**

```
Given I have built a family tree
When I open the export modal and choose “JSON”
Then a file named projectname-YYYYMMDD.json is downloaded containing
all persons, relationships, and events according to the schema
```

Scope (MVP vs Post-MVP)

MVP includes:

1. Local-first SPA functioning offline.
2. Create, edit and delete persons and relationships (biological, adoptive, step/half, partners/spouses).
3. Approximate date entry using about/before/after/between [1].
4. Undo/redo with unlimited history (until page reload).
5. Export to JSON and CSV.
6. Simple import of previously exported JSON.
7. Accessibility features meeting WCAG 2.2 AA (focus order, keyboard navigable, drag alternative [2] [3]) and support for screen readers.
8. Internationalization (en/es/pt) with locale-appropriate name and date formats.

Post-MVP enhancements:

1. Cloud sync and collaboration via optional account.
2. GEDCOM export (limited mapping due to format restrictions).
3. Import from GEDCOM with best-effort mapping.
4. Multimedia attachments (photos, documents).
5. Printing/exporting to PDF/SVG and custom layouts.
6. AI-assisted suggestions (e.g., duplicate detection, date estimation).

Non-goals:

1. Real-time multi-user editing (reserved for later iteration).
2. Public search or genealogical record lookups (beyond scope).
3. Full support of every GEDCOM nuance; only the commonly used subset is mapped.
4. Complex research features such as evidence analysis or citations management.

Success Metrics

Metric	Definition
Activation	% of new visitors who create a tree and add at least one person.
Task success	Median time to add two generations of a family.
Export completion	% of trees successfully exported to JSON/CSV.
Error rate	Average number of unexpected errors per session; aim to keep below 2%.

Metric	Definition
Accessibility compliance	Automated checks passing WCAG 2.2 AA; manual audits confirming navigation order and focus not obscured [2] [3] .
International usage	Number of sessions using non-English locale and satisfaction scores.

UX Specification

Information Architecture and Navigation

The application is composed of the following major sections:

1. **Home/Projects page:** displays a list of existing trees (if saved locally) and a button to create a new tree. When cloud sync is enabled, it also lists remote projects.
2. **Canvas/Editor:** the main workspace where nodes (persons) are displayed as cards on a canvas. A top navigation bar contains project name, undo/redo, zoom controls, export/import, and settings. A left side panel (drawer) shows details of the selected person or relationship.
3. **Person Detail Panel:** reveals when a person is selected. Contains fields for name (given, surname, suffix), sex/gender, pronouns, events (birth, death, marriage), notes, sources and media references. Allows toggling approximate date modes and marking the person as deceased.
4. **Relationship Manager:** accessible via context menu; allows changing relationship type (e.g., adoptive, step), adding spouses or partners, and merging duplicates.
5. **Export/Import Modal:** provides options for JSON, CSV and GEDCOM export; shows file size and version. Import tab accepts JSON files to restore a tree.
6. **Settings:** allows switching language, date format, theme (light/dark/high contrast), enabling/disabling cloud sync, clearing local storage and reading privacy policy.

User Flows

Below are high-level user flows represented in Mermaid diagrams. The flows include alternate paths to support keyboard navigation (click-to-connect) in addition to drag-and-drop.

Create Tree Flow

```
flowchart TD
    A(Home page) --> B{Existing project?}
    B -- No --> C[Create New Tree]
    C --> D[Canvas with root person]
    B -- Yes --> E[Select project]
    E --> D
```

Add Person and Relationship Flow

```
flowchart TD
    S(Select existing person) --> K{Add mode?}
    K -- Parent --> P[Open Person Form]
    P --> R[Create new person]
    R --> R2[Link as parent (type biological, adoptive, step)]
    K -- Child --> C1
    C1[Open Person Form] --> C2[Create new person] --> C3[Link as child]
    K -- Spouse/Partner --> M1[Open Person Form] --> M2[Create new person] -->
M3[Link as spouse/partner]
    K -- Sibling --> SI[Open Person Form] --> SJ[Create new person] --> SK[Link
as sibling]
    K -- Cancel --> S
```

Edit Details & Approximate Dates

```
flowchart TD
    X(Select person) --> Y[Open detail panel]
    Y --> Z{Edit section?}
    Z -- Name --> Z1[Edit name fields]
    Z -- Sex --> Z2[Edit sex/gender/pronouns]
    Z -- Events --> E1
    E1[Select event] --> E2{Exact or Approx?}
    E2 -- Exact --> E3[Enter full date]
    E2 -- Approximate --> E4[Choose type: About/Before/After/Between] -->
E5[Enter value(s)]
    Z -- Notes --> Z3[Edit notes/sources]
    Z -- Delete person --> Z4[Confirm deletion]
```

Merge Duplicates Flow

```
flowchart TD
    M(Select person) --> M1[Open context menu]
    M1 --> M2[Choose “Merge”]
    M2 --> M3[Select duplicate person]
    M3 --> M4[Resolve conflicts (choose preferred values)]
    M4 --> M5[Combined person saved]
```

Export Flow

```
flowchart TD
    E0(Open export modal) --> E1{Format?}
```

```
    E1 -- JSON --> J[Serialize tree to JSON]
    J --> D1[Download JSON file]
    E1 -- CSV --> V[Generate persons.csv, relationships.csv, events.csv]
    V --> D2[Download ZIP of CSV files]
    E1 -- GEDCOM (post-MVP) --> G[Convert tree to GEDCOM 5.5.5 format]
    G --> D3[Download .ged file]
```

ASCII Wireframes

The wireframes below illustrate the layout for key screens. They are not pixel-perfect but show the information hierarchy.

Home/Start Screen

```
┌──────────────────────────────────────────────┐
│           PROJECT_NAME Home                  │
├──────────────────────────────────────────────┤
│ [Create New Tree]                            │
│                                              │
│ Saved Trees:                                 │
│ ┌─────────────┐ ┌─────────────┐              │
│ │ Family A    │ │ Family B    │ ...          │
│ └─────────────┘ └─────────────┘              │
│                                              │
│ Settings  |  Language  |  Help               │
└──────────────────────────────────────────────┘
```

Canvas/Editor

```
┌──────────────────────────────────────────────────────────┐
│ Project Name   Undo/Redo  Zoom – +  Export  Settings     │
├──────────────────────────────────────────────────────────┤
│   ••• Canvas (panning & zooming)                         │
│   [Node]──[Node]                                         │
│     │        │                                           │
│     ▼        ▼                                           │
│   [Node]   [Selected Node]-> Detail Panel on left        │
│                                                          │
├──────────────────────────────────────────────────────────┤
│ Status bar: “Use mouse to drag/pan or select a node”     │
└──────────────────────────────────────────────────────────┘
```

Person Detail Panel

```
┌──────────────────────┐
│ Person Details       │
├──────────────────────┤
│ Name: [Given] [Surname] [Suffix]      │
│ Sex: [Male/Female/Non-binary/Custom]  │
│ Pronouns: [he/she/they/custom]        │
│ Events:                                │
│   • Birth: [Date picker] [Location]    │
│   • Death: [Date picker] [Location]    │
│   • Marriage(s): [Add]                 │
│ Notes / Sources: [textarea]            │
│ Media: [upload button]                 │
│ [Delete Person] [Merge…]               │
└──────────────────────┘
```

Relationship Manager

```
┌──────────────────────────────┐
│ Relationship Manager         │
├──────────────────────────────┤
│ Person A — [Type] — Person B  │
│ Type: [Biological / Adoptive / Step / Partner / Sibling]  │
│ Notes: [textarea]          │
│ [Save] [Cancel]            │
└──────────────────────────────┘
```

Export Modal

```
┌──────────────────────────────┐
│ Export / Import              │
├──────────────────────────────┤
│ Format: ( ) JSON  ( ) CSV  ( ) GEDCOM* │
│ Version: v1.0 (JSON schema) │
│ Include media: [ ]          │
│ [Export]                    │
├──────────────────────────────┤
│ Import JSON: [Choose File] │
│ [Import]                    │
└──────────────────────────────┘
* GEDCOM available post-MVP
```

Interaction Details

Drag-to-connect: Users can drag a connector from one node to another to create relationships. To support keyboard and accessibility requirements, the same operation can be completed by selecting a node, pressing a dedicated key (e.g., "P" for parent), and choosing the target person from a list, fulfilling WCAG 2.2's alternative to dragging [3].

Context menus: Right-click (or long-press on mobile) on a node opens a menu with actions: Add Parent/Child/Spouse/Sibling, Edit, Delete, Merge, and Manage Relationships. On mobile, the menu appears after a tap-hold gesture.

Keyboard shortcuts:

Action	Shortcut
Undo	Ctrl+Z
Redo	Ctrl+Shift+Z
Zoom in/out	Ctrl+ ±
Fit to screen	F
Select parent/child/spouse mode	P/C/S
Toggle left panel	Tab

Mobile gestures: Pinch to zoom; double-tap to center; long-press for context menu; drag with one finger to pan; tap on plus buttons to add nodes.

Zoom/Pan: Canvas supports smooth panning (mouse drag or arrow keys) and pinch-to-zoom. Fit-to-screen button centers the full tree. A minimap (bottom right) shows the entire tree with a viewport indicator.

Snapping & Auto-layout: Nodes snap to a grid by default; users can toggle auto-layout to automatically arrange nodes using a tree layout algorithm. Auto-layout can be run on demand rather than continuously to preserve manual adjustments.

Accessibility (WCAG 2.2 AA) Considerations

1. **Keyboard Navigation:** All actions must be accessible via keyboard. Focus order must preserve meaning [2]; e.g., the focus should move logically from the toolbar to the canvas and then to the detail panel.
2. **Focus Not Obscured:** When an element receives focus, it must remain visible and not be covered by other UI elements [3]. Implementation should auto-scroll to keep focused nodes in view.
3. **Focus Appearance:** Provide a clear focus indicator with sufficient contrast (≥3:1) [3].
4. **Alternative to Dragging:** Provide keyboard alternatives for drag operations [3].
5. **Target Size:** Interactive elements must be at least 24×24 px to aid users with limited dexterity [3].

6. **ARIA Roles:** Represent nodes as `role="button"` or `role="group"` with appropriate labels; connections have `role="link"`. The canvas should be a `role="application"` with instructions for screen reader users.
7. **Screen Reader Descriptions:** Provide textual descriptions of relationships (e.g., “John Doe, father of Jane Doe”) via `aria-label` so graph semantics are communicated verbally.
8. **Colour and Contrast:** Use design tokens to ensure high contrast; support dark/high-contrast modes.

Internationalization Plan

1. **Languages:** Support English (default), Spanish, and Portuguese via a translation library (e.g., i18next). Externalize all user-visible strings.
2. **Name Order:** Allow customizing display order (Given Name–Surname or Surname–Given Name) since Spanish and Portuguese often include multiple surnames.
3. **Date Localization:** Use locale-aware date pickers; support day-month-year and month-day-year formats; approximate date labels must also be localized (“Abt.” for English, “c.” for Spanish “aprox.” etc.).
4. **RTL Considerations:** Provide mirrored layouts and direction support; not high priority for initial languages but ensure code does not assume left-to-right only.
5. **Pluralization and Gendered Language:** Use pluralization rules; avoid gendered pronouns in UI; allow user-specified pronouns in person cards.

Data & Domain Model

Concepts & Entity Definitions

The genealogical domain comprises several core concepts:

1. **Tree/Project:** A container for all persons, relationships, events, sources and media for a particular family tree.
2. **Person:** An individual with a unique identifier. Fields include:
3. `id` (UUID or opaque string)
4. `names`: array of objects with `given`, `surname`, `suffix`, `preferred` (boolean) and `full` (cached full name)
5. `sex`: enumeration {“male”, “female”, “non-binary”, “unknown”, “custom”}
6. `gender`: free-form string or pronouns
7. `birth`, `death`: event references
8. `events`: array of event IDs (birth, death, marriage, adoption, etc.)
9. `notes`: array of note IDs
10. `sources`: array of source IDs
11. `media`: array of media IDs
12. `customFields`: key-value pairs for user-defined attributes
13. **Relationship:** Connects two or more persons. Fields:

14. `id`
15. `type`: enumeration such as `biologicalParentChild`, `adoptiveParentChild`, `stepParent`, `partner`, `marriage`, `divorce`, `sibling`, `halfSibling`, `guardian`, `unknown`
16. `persons`: array of person IDs participating in the relationship (with optional roles to distinguish parent vs. child)
17. `startEventId`, `endEventId`: references to events (e.g., marriage and divorce)
18. `notes`, `sources`, `media`

19. **Event:** Describes life events. Fields:

20. `id`
21. `type`: enumeration {`birth`, `death`, `marriage`, `adoption`, `divorce`, `custom`}
22. `date`: object with `type` (`exact`, `about`, `before`, `after`, `between`) and `value` (ISO date string or array of two dates)
23. `place`: free-form string or geo reference
24. `description`
25. `notes`, `sources`, `media`

26. **Source/Note:** Records references to documents or notes. Fields: `id`, `title`, `text`, `citation`, `url`, `type` (e.g., book, record), `media` references.

27. **Media:** Metadata about images or documents; actual binary content may be stored as data URLs or separate files. Fields: `id`, `filename`, `mimeType`, `size`, `dateAdded`, `description`.

Relationship Modeling Notes

The system must support a variety of relationships beyond traditional nuclear families:

1. **Biological Parent–Child**: A pair of persons with a natural parent relationship.
2. **Adoptive Parent–Child**: Similar to biological but flagged as adoptive. Step-parents and guardians can be represented using separate relationship types with optional start and end events.
3. **Marriage/Partnership**: Connects two persons as spouses or partners; same-sex relationships are supported. GEDCOM uses `FAM` records with `HUSB` and `WIFE` tags but clarifies that gender roles should not be inferred [4]. Our model treats partners as symmetric with optional roles such as “partner1” and “partner2”.
4. **Divorce/Annulment**: Relationship with an end event referencing a divorce or annulment event.
5. **Step/Half Siblings**: Derived relationships. When a person shares one parent but not both, the system designates them as half siblings. When a parent is married to someone who is not the child’s biological parent, a step relationship is added (common in blended families [5]).
6. **Unknown/Uncertain**: Allows connecting persons with uncertain relationships (e.g., unknown father). These relationships can be marked as `unknownParentChild` and updated later when more information is available.

Constraints & Validations

1. **No Cycles:** Prevent a person from becoming their own ancestor or descendant. Graph traversal prevents adding relationships that would create a cycle.
2. **Approximate Dates:** Use `type` (`about`, `before`, `after`, `between`) and store ISO-8601 strings or ranges; ensure `between` includes a start ≤ end.
3. **Duplicate Detection:** Suggest duplicates based on matching names and overlapping dates. Provide merge flow to resolve conflicts.
4. **Gender & Pronouns:** Support unspecified or non-binary values; avoid inferring gender from relationship roles. GEDCOM historically uses binary roles but explicitly states that `HUSB` / `WIFE` tags do not denote gender [4].
5. **Event Ordering:** Birth must precede death. Relationship start events must not occur before participant birth dates.
6. **Person Deletion:** Deleting a person must remove associated relationships and events or prompt the user to reassign relationships.

Entity Relationship Diagram (ERD)

The following Mermaid ERD illustrates the relationships between entities.

```
erDiagram
    TREE ||--o{ PERSON : contains
    TREE ||--o{ RELATIONSHIP : contains
    TREE ||--o{ EVENT : contains
    TREE ||--o{ SOURCE : contains
    TREE ||--o{ MEDIA : contains
    PERSON ||--o{ EVENT : has
    PERSON ||--o{ RELATIONSHIP : participates
    RELATIONSHIP ||--o{ EVENT : hasStart
    RELATIONSHIP ||--o{ EVENT : hasEnd
    EVENT ||--o{ SOURCE : cites
    EVENT ||--o{ MEDIA : uses
    PERSON ||--o{ SOURCE : cites
    PERSON ||--o{ MEDIA : uses
    RELATIONSHIP ||--o{ SOURCE : cites
    RELATIONSHIP ||--o{ MEDIA : uses
    SOURCE ||--o{ MEDIA : includes
```

JSON Schema 2020-12

Below is a simplified version of the JSON Schema for exported trees. The schema is versioned using `$id` and `$schema` so that future changes can evolve without breaking compatibility.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/schemas/project-tree-1.0.json",
```

```
  "title": "Genealogical Tree Export",
  "type": "object",
  "properties": {
    "schemaVersion": { "type": "string", "const": "1.0" },
    "projectId": { "type": "string", "format": "uuid" },
    "projectName": { "type": "string" },
    "created": { "type": "string", "format": "date-time" },
    "persons": {
      "type": "array",
      "items": { "$ref": "#/definitions/person" }
    },
    "relationships": {
      "type": "array",
      "items": { "$ref": "#/definitions/relationship" }
    },
    "events": {
      "type": "array",
      "items": { "$ref": "#/definitions/event" }
    },
    "sources": {
      "type": "array",
      "items": { "$ref": "#/definitions/source" }
    },
    "media": {
      "type": "array",
      "items": { "$ref": "#/definitions/media" }
    }
  },
  "required": ["schemaVersion", "projectId", "projectName", "persons",
"relationships"],
  "definitions": {
    "person": {
      "type": "object",
      "properties": {
        "id": { "type": "string" },
        "names": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "given": { "type": "string" },
              "surname": { "type": "string" },
              "suffix": { "type": "string" },
              "preferred": { "type": "boolean" },
              "full": { "type": "string" }
            },
            "required": ["given", "surname", "preferred"]
          }
```

```
        },
        "sex": { "type": "string", "enum": ["male", "female", "non-binary",
"unknown", "custom"] },
        "gender": { "type": "string" },
        "events": { "type": "array", "items": { "type": "string" } },
        "notes": { "type": "array", "items": { "type": "string" } },
        "sources": { "type": "array", "items": { "type": "string" } },
        "media": { "type": "array", "items": { "type": "string" } },
        "customFields": { "type": "object", "additionalProperties": true }
      },
      "required": ["id", "names"]
    },
    "relationship": {
      "type": "object",
      "properties": {
        "id": { "type": "string" },
        "type": {
          "type": "string",
          "enum": [
            "biologicalParentChild", "adoptiveParentChild", "stepParentChild",
            "partner", "marriage", "divorce", "sibling", "halfSibling",
            "guardian", "unknown"
          ]
        },
        "persons": { "type": "array", "items": { "type": "string" },
"minItems": 2 },
        "startEventId": { "type": ["string", "null"] },
        "endEventId": { "type": ["string", "null"] },
        "notes": { "type": "array", "items": { "type": "string" } },
        "sources": { "type": "array", "items": { "type": "string" } },
        "media": { "type": "array", "items": { "type": "string" } }
      },
      "required": ["id", "type", "persons"]
    },
    "event": {
      "type": "object",
      "properties": {
        "id": { "type": "string" },
        "type": { "type": "string", "enum": ["birth", "death", "marriage",
"adoption", "divorce", "custom"] },
        "date": {
          "type": "object",
          "properties": {
            "type": { "type": "string", "enum": ["exact", "about", "before",
"after", "between"] },
            "value": {
              "oneOf": [
                { "type": "string", "format": "date" },
```

```
                { "type": "array", "items": { "type": "string", "format":
"date" }, "minItems": 2, "maxItems": 2 }
              ]
            }
          },
          "required": ["type", "value"]
        },
        "place": { "type": "string" },
        "description": { "type": "string" },
        "notes": { "type": "array", "items": { "type": "string" } },
        "sources": { "type": "array", "items": { "type": "string" } },
        "media": { "type": "array", "items": { "type": "string" } }
      },
      "required": ["id", "type", "date"]
    },
    "source": {
      "type": "object",
      "properties": {
        "id": { "type": "string" },
        "title": { "type": "string" },
        "text": { "type": "string" },
        "citation": { "type": "string" },
        "url": { "type": "string", "format": "uri" },
        "type": { "type": "string" },
        "media": { "type": "array", "items": { "type": "string" } }
      },
      "required": ["id", "title"]
    },
    "media": {
      "type": "object",
      "properties": {
        "id": { "type": "string" },
        "filename": { "type": "string" },
        "mimeType": { "type": "string" },
        "size": { "type": "integer" },
        "dateAdded": { "type": "string", "format": "date-time" },
        "description": { "type": "string" }
      },
      "required": ["id", "filename", "mimeType"]
    }
  }
}
```

Example Export File (JSON)

```
{
  "schemaVersion": "1.0",
  "projectId": "123e4567-e89b-12d3-a456-426614174000",
  "projectName": "Doe Family",
  "created": "2025-08-07T12:34:56Z",
  "persons": [
    {
      "id": "p1",
      "names": [
        {"given": "John", "surname": "Doe", "suffix": "", "preferred": true,
"full": "John Doe" }
      ],
      "sex": "male",
      "gender": "he/him",
      "events": ["e1", "e2"],
      "notes": [],
      "sources": [],
      "media": []
    },
    {
      "id": "p2",
      "names": [
        {"given": "Jane", "surname": "Smith", "suffix": "", "preferred": true,
"full": "Jane Smith" }
      ],
      "sex": "female",
      "gender": "she/her",
      "events": ["e3"],
      "notes": [],
      "sources": [],
      "media": []
    }
  ],
  "relationships": [
    {
      "id": "r1",
      "type": "marriage",
      "persons": ["p1", "p2"],
      "startEventId": "e3",
      "endEventId": null,
      "notes": [],
      "sources": [],
      "media": []
    },
    {
```

```
      "id": "r2",
      "type": "biologicalParentChild",
      "persons": ["p1", "p3"],
      "startEventId": null,
      "endEventId": null,
      "notes": [],
      "sources": [],
      "media": []
    }
  ],
  "events": [
    {
      "id": "e1",
      "type": "birth",
      "date": {"type": "exact", "value": "1950-04-01"},
      "place": "Springfield, USA",
      "description": "Birth of John Doe",
      "notes": [],
      "sources": [],
      "media": []
    },
    {
      "id": "e2",
      "type": "death",
      "date": {"type": "about", "value": "2020"},
      "place": "Unknown",
      "description": "Approximate death year",
      "notes": [],
      "sources": [],
      "media": []
    },
    {
      "id": "e3",
      "type": "marriage",
      "date": {"type": "exact", "value": "1975-06-20"},
      "place": "City Hall",
      "description": "Marriage of John and Jane",
      "notes": [],
      "sources": [],
      "media": []
    }
  ],
  "sources": [],
  "media": []
}
```

Mapping to GEDCOM (Optional Post-MVP)

GEDCOM is a plain text format for genealogical data exchange [6]. The widely used version is 5.5.5 (2019) [7]; 5.5.1 (1999/2019) remains common [8]. GEDCOM's lineage-linked model uses `INDI` records for individuals and `FAM` records for families; relationships are encoded through `HUSB`, `WIFE`, `CHIL` tags, but the specification warns not to infer gender from these labels [4]. Our mapping must respect this nuance.

Internal Concept	GEDCOM Tag	Notes
Person name	`NAME` with given name and surname separated by `/`	GEDCOM supports multiple NAME tags.
Sex	`SEX`	Use `M` / `F` or `U` for unknown; custom genders may be stored in notes as not directly supported.
Birth event	`BIRT` followed by `DATE` and `PLAC`	Approximate dates use GEDCOM qualifiers (e.g., `ABT`, `BEF`, `AFT`).
Death event	`DEAT` record	Similar to birth.
Marriage event	`FAM` record with `MARR` under it; `HUSB` and `WIFE` link individuals	For same-sex partnerships, both persons may be placed in HUSB/WIFE positions; use note explaining gender neutrality [4].
Divorce	`DIV` under `FAM`	
Parent-child	`FAM` record's `CHIL` links children	Additional attributes (adoptive, step) may be stored as custom tags (e.g., `_ADOP`).
Notes	`NOTE` records and references	
Sources	`SOUR` records	
Media	`OBJE` records or GEDZip (v7)	GEDCOM 7 introduced GEDZip; older versions store only references [8].

Limitations: GEDCOM's binary gender tags hinder full representation of non-binary individuals. Some relationship types (guardianship, step) are not standardized; custom tags (`_GUAR`) may be used but compatibility varies. Also, GEDCOM 7's UTF-8 requirement may not be supported by all software [8].

Export/Import Specification

Export Formats

1. **JSON (canonical)** – uses the schema defined above. File naming convention: `<projectName>-<YYYYMMDD>.json` and optionally `<projectName>-<YYYYMMDD>.json.gz` for compressed files. The exported file is self-contained and includes schema version information.

2. **CSV** – exports tables for persons, relationships and events. Each file includes headers and IDs to link across tables. For example:

persons.csv

id	given	surname	sex	gender
p1	John	Doe	male	he/him

relationships.csv

id	type	person1	person2	startEventId	endEventId
r1	marriage	p1	p2	e3	

events.csv

id	type	dateType	dateValue	place	description
e3	marriage	exact	1975-06-20	City Hall	Marriage of John and Jane

The CSV export is packaged as a ZIP to keep files together. Column order and names should remain stable across versions.

1. **GEDCOM** (post-MVP) – generate a `.ged` file following GEDCOM 5.5.5. Provide version in header (e.g., `1 GEDC VER 5.5.5`). Use UTF-8 encoding and line breaks per specification [9]. Limitations noted above.

Import Specification

1. **JSON Import:** Accepts JSON files conforming to the schema. Validate `schemaVersion`; if unknown, warn the user. Map persons, relationships and events by ID. For conflicts (duplicate IDs), prompt the user to merge or create new IDs. Provide a dry-run preview summarizing how many persons and relationships will be imported.

2. **CSV Import:** (Optional) Accepts a ZIP of CSV files with correct headers. Parse files and construct objects. Validate cross-references; if unmatched IDs exist, error out and list them.

3. **GEDCOM Import:** (Post-MVP) Parse GEDCOM to extract individuals (`INDI`), families (`FAM`), events (`EVEN`), notes and sources. Mapping is best-effort due to GEDCOM's constraints: unknown genders become `unknown`, custom relationships may be lost. Provide warnings for unsupported tags.

Compression Options

For large trees, JSON export may compress significantly. Provide an option to download files with `.gz` compression using the browser's compression library. The import UI must transparently decompress `.gz` files.

System Architecture & Tech Choices

Build Track A – Local-First SPA

Rationale: A local-first Single Page Application (SPA) ensures privacy and offline capability by storing all data in the user's browser (IndexedDB). This approach avoids the need for servers, reduces compliance overhead, and offers fast interactions. It aligns with the requirement that no account is necessary.

Technologies

Layer	Option	Decision Notes
UI Framework	**React** with Vite or SvelteKit	React has a mature ecosystem, robust accessibility tooling and virtualization libraries; SvelteKit offers smaller bundle sizes and built-in SSR for later cloud variant.
State Management	Redux Toolkit or Zustand	Centralized store with time-travel debugging simplifies undo/redo and offline persistence.
Storage	IndexedDB via a wrapper (e.g., Dexie.js)	Persistent offline storage for persons, relationships, events; supports large datasets.
Graph Rendering	SVG for small to medium trees; fallback to Canvas/WebGL via libraries like PixiJS for >3k nodes	Canvas/WebGL handles thousands of nodes efficiently.
Layout Algorithm	Reingold–Tilford tree layout (for hierarchical view); custom radial layout for ancestors/ descendants	Provide switchable layouts for user preference.
Language & Type	TypeScript	Improves reliability and maintainability.

High-Level Component Diagram

```
graph TD
  UI[User Interface (React/Svelte)] -->|dispatch| Store[State Store]
  Store -->|persist| IndexedDB
  UI --> Layout[Layout Engine]
  Layout --> CanvasRenderer[Renderer (SVG/Canvas/WebGL)]
  UI --> Exporter[Export & Import Module]
  Exporter --> FileSystem[Browser FileSystem API]
```

Data Flow

1. User interaction triggers actions (e.g., add person).
2. Actions are dispatched to the state store; reducers update the in-memory state.
3. Middleware persists changes to IndexedDB and updates the undo/redo stack.
4. UI listens to state changes and re-renders the canvas via the layout engine.
5. Exporter serializes the store to JSON/CSV and triggers a download.

Offline Sync (Future Cloud Variant)

Although local-first, the architecture should allow eventual synchronization. When cloud sync is enabled, an offline-first database (e.g., PouchDB) with replication to a CouchDB/Cloud Firestore backend can be used. Conflict resolution relies on a Last-Write-Wins strategy or CRDTs; merging duplicates remains manual due to genealogical nuance.

Build Track B – Cloud-Enabled Variant

Rationale: Some users may desire backups and multi-device access. The cloud track introduces an API and authentication layer while preserving offline capability via service workers and background sync. Data encryption and secure transmission are mandatory given the sensitive nature of genealogical data (birth dates, family relationships). The OWASP cryptographic storage cheat sheet recommends encrypting data at the application or database level and using AES-128/256 or ECC [10]; TLS 1.3 should be the default transport protocol [11].

Technologies

Layer	Option	Decision Notes
Backend Framework	Next.js with API routes (Node) or FastAPI (Python)	Next.js integrates seamlessly with React front-end; FastAPI offers strong typing and performance.
Database	PostgreSQL or MongoDB	Stores projects, persons, relationships; PostgreSQL ensures relational integrity, but MongoDB may be chosen for schema flexibility.
Authentication	OAuth 2.0 (Google, email/password) with JWTs	Allows optional accounts; tokens used for API requests.
Hosting	Vercel/Netlify for front-end; serverless functions for API (AWS Lambda)	Simplifies deployment and scaling.
Encryption	Use TLS 1.3 for all API endpoints [11]; at-rest encryption via managed database services.	

High-Level Component Diagram (Cloud)

```
graph TD
  BrowserApp -->|REST/GraphQL| API[Backend API]
  API --> Auth[Auth Service]
  API --> DB[(Database)]
  API --> Storage[(Object Storage for media)]
  API --> ExportService
  BrowserApp --> ServiceWorker[Service Worker & IndexedDB]
  ServiceWorker -->|Sync| API
```

Data Flow (Cloud)

1. User signs in (optional). JWT token stored in browser.
2. Local actions update IndexedDB; a sync service queues changes.
3. When online, the service worker sends batched changes to the API; the API authenticates the request and writes to the database.
4. The API responds with updated state; conflicts are resolved by timestamps or CRDT algorithms.
5. Export and import endpoints allow server-side processing (e.g., GEDCOM conversion).

Performance Plan

1. **Rendering large trees:** Use virtualization (only render visible nodes); for very large graphs, switch to Canvas/WebGL. Use requestAnimationFrame for smooth panning/zooming. Maintain frame rates above 60 FPS.
2. **Layout computation:** Offload to Web Workers to prevent blocking the UI. Precompute positions for static branches. Provide incremental layout updates when adding nodes.
3. **Core Web Vitals:** Target Largest Contentful Paint < 2.5 s, First Input Delay < 100 ms, Cumulative Layout Shift near 0. Preload fonts and lazy-load heavy modules (e.g., GEDCOM exporter).

Security & Privacy

1. **Threat Model:** Data includes personally identifiable information (names, birth dates). Primary threats are unauthorized access (if cloud), data leakage via cross-site scripting, insecure storage and transmission.
2. **Transport Security:** Use HTTPS with TLS 1.3; disable older protocols; use strong ciphers [11].
3. **Encryption at Rest:** When syncing to cloud, encrypt sensitive fields in the database (e.g., using AES-256) [10]. For local storage, the browser sandbox mitigates many threats; provide an option to encrypt export files with a passphrase.
4. **Authentication & Authorization:** Use JWT with short expiry; refresh tokens securely; enforce rate limits. Use role-based access control (owner vs collaborator).
5. **Content Security Policy (CSP):** Restrict script, style and media sources to trusted domains; mitigate XSS.
6. **Backup & Restore:** Provide versioned backups for cloud data; allow user-initiated exports. Local data can be backed up via export.
7. **GDPR/Privacy:** Default to local-only storage; no tracking without opt-in. Provide clear privacy notice; allow deletion of all data.

Scalability

The local-first architecture scales per device, limited by IndexedDB capacity (hundreds of MB). The app is designed to handle trees with thousands of nodes; virtualization and offloading of layout computation ensure responsiveness. The cloud backend can scale horizontally via serverless architecture; database partitioning by user ensures independent scaling. In the future, support for shared public trees may require additional caching and search services.

API Specification (Cloud Variant)

Although the MVP does not include cloud sync, we provide a stubbed OpenAPI 3.1 specification to guide future development. Endpoints are namespaced under `/api/v1/`. Authentication uses Bearer JWT tokens. Pagination uses `limit` and `offset` query parameters. Errors follow RFC 7807 (“problem details”) with JSON bodies containing `type`, `title`, `status`, `detail`, and `instance`.

```
openapi: 3.1.0
info:
  title: PROJECT_NAME API
  version: 0.1.0
servers:
  - url: https://api.projectname.example.com/api/v1
components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
  schemas:
    Person:
      $ref: '#/components/examples/PersonExample'
    Error:
      type: object
      properties:
        type: { type: string, format: uri }
        title: { type: string }
        status: { type: integer }
        detail: { type: string }
        instance: { type: string, format: uri }
      required: [title, status]
  examples:
    PersonExample:
      value:
        id: p1
        names:
          - given: John
```

```
            surname: Doe
            preferred: true
        sex: male
        gender: he/him
        events: []
        notes: []
        sources: []
        media: []
security:
  - bearerAuth: []
paths:
  /projects:
    get:
      summary: List projects
      responses:
        '200':
          description: A list of projects
    post:
      summary: Create a project
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                name: { type: string }
              required: [name]
      responses:
        '201':
          description: Project created
  /projects/{projectId}:
    get:
      summary: Get a project
      parameters:
        - in: path
          name: projectId
          required: true
          schema: { type: string }
      responses:
        '200': { description: Project details }
  /projects/{projectId}/persons:
    get:
      summary: List persons
    post:
      summary: Create a person
      requestBody:
        required: true
```

```
      content:
        application/json:
          schema: { $ref: '#/components/schemas/Person' }
    responses:
      '201': { description: Person created }
/projects/{projectId}/relationships:
  get:
    summary: List relationships
  post:
    summary: Create a relationship
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
    responses:
      '201': { description: Relationship created }
/projects/{projectId}/export:
  get:
    summary: Export a project
    parameters:
      - in: path
        name: projectId
        required: true
        schema: { type: string }
      - in: query
        name: format
        schema: { type: string, enum: [json, csv, gedcom] }
    responses:
      '200': { description: Export file }
/projects/{projectId}/import:
  post:
    summary: Import data into a project
    requestBody:
      required: true
      content:
        application/json:
          schema: { type: object }
    responses:
      '202': { description: Import accepted }
```

The stub can be expanded with additional endpoints (e.g., authentication, media upload) as the cloud variant is developed.

Front-End Component Library & UI Spec

Component List

Component	Purpose	Props & Events
TreeCanvas	Renders nodes and edges; supports panning/zooming, selection, drag-and-drop, minimap.	Props: `persons`, `relationships`, `selectedId`, `layoutMode`; Events: `onSelect(id)`, `onAddConnection(sourceId, targetId, type)`, `onPan(offset)`, `onZoom(scale)`.
NodeCard	Visual representation of a person; shows name(s), icons (deceased, unknown), and relationship handles.	Props: `person`, `isSelected`, `position`; Events: `onClick`, `onContextMenu`.
RelationConnector	Interactive handles for connecting nodes; appears on NodeCard edges.	Props: `direction` (parent/child/spouse/sibling), `onStartConnection`.
PersonForm	Form for editing person details in the detail panel.	Props: `person`, `onChange(field, value)`, `onDelete`, `onMerge`.
RelationshipForm	Form for editing relationship type and notes.	Props: `relationship`, `onChange(field, value)`, `onDelete`.
DatePickerApprox	Custom date picker supporting approximate types.	Props: `value`, `onChange`, `locale`; Renders drop-down for type and appropriate input fields.
ExportDialog	Modal for selecting export format and initiating download.	Props: `open`, `onClose`, `onExport(format, options)`.
ImportDialog	Modal for selecting a file to import.	Props: `open`, `onClose`, `onImport(file)`.
ProjectList	Home page list of existing projects; supports create/delete.	Props: `projects`, `onSelect`, `onCreate`, `onDelete`.
SettingsPanel	Allows users to change language, theme, enable sync.	Props: `settings`, `onChange`.

Component	Purpose	Props & Events
Toast	Small notifications for success/error.	Props: `message`, `type`, `duration`.
Minimap	Overview of the tree; click to pan.	Props: `persons`, `relationships`, `viewport`; Event: `onClick(position)`.
Toolbar	Top bar containing project name, undo/redo buttons, zoom controls, export/import, settings.	Props: `projectName`, `undoDisabled`, `redoDisabled`; Events: `onUndo`, `onRedo`, `onZoomIn`, `onZoomOut`, `onFit`, `onExport`, `onImport`, `onSettings`.

State Management & Undo/Redo

The application uses a centralized store (Redux/Zustand) with normalized entities (persons, relationships, events). The undo/redo stack stores a list of patches (inverse operations). When an action occurs (e.g., add person), a patch is generated (e.g., remove that person) and pushed onto the undo stack. Redo uses the original action again. History persists across sessions using IndexedDB. Middleware debounces saves to avoid performance degradation.

Autosave Strategy

Autosave occurs automatically after each commit to the state store. When offline, data persists in IndexedDB; when cloud sync is enabled, changes are queued and synchronized in the background. Users may also manually download backups via export.

Theming & Design Tokens

Implement a design system with tokens for colors, typography, spacing and border radii. Provide at least three themes: light (default), dark, and high-contrast. Use CSS custom properties and let users switch themes in settings. The same token definitions apply across React or Svelte components. Provide accessible contrasts (≥4.5:1 for body text).

Validation, Testing & Quality

Domain Validation Rules

Rule	Rationale	Example
A person cannot be their own ancestor or descendant	Prevents impossible cycles in the family graph	When adding a parent, ensure the parent is not already a descendant of the child.

Rule	Rationale	Example
Dates must be valid and approximate ranges must be chronological	Ensures data integrity	“Between 1870 and 1880” cannot have the second date earlier than the first.
Birth must occur before death	Maintains chronology	Reject a death date earlier than birth date.
Only two parents (biological/adoptive) may be linked for a child in most jurisdictions; additional guardians can be added as separate relationships	Reflects typical genealogical conventions and simplifies layout	Adding a third adoptive parent triggers a warning.
Relationship types must correspond to allowed enumerations	Avoids invalid data	Prevent entering arbitrary strings in relationship type.
Name fields cannot be all empty	Ensures a person has at least one name for identification.	

Testing Strategy

1. **Unit Tests:** Test reducers/actions for state management; validate date parsing and approximate date logic; test components with shallow rendering; use Vitest or Jest.
2. **Integration Tests:** Simulate user flows (create person, link relationship) in the browser using React Testing Library. Test asynchronous behavior like autosave and import/export.
3. **End-to-End Tests:** Use Playwright or Cypress to test flows across the UI: create tree, add nodes via drag and keyboard, export file and verify contents. Include tests for accessibility (axe-core) to catch violations.
4. **Schema Tests:** Validate exported JSON against the JSON Schema using AJV. Write tests to ensure exports remain compatible when schema version increments.
5. **Accessibility Linting:** Integrate accessibility linters (eslint-plugin-jsx-a11y) and run axe tests during CI.
6. **Performance & Size Budgets:** Use lighthouse to monitor core web vitals; enforce bundle size budgets; run regression tests for layout performance with thousands of nodes.
7. **Cross-Browser & Mobile Testing:** Ensure the application functions on Chrome, Firefox, Safari and mobile browsers; test responsive layouts with various breakpoints.

Example Unit Test (Pseudo-code)

```
test('adding a parent creates new person and relationship', () => {
  const { store } = setupTestStore();
  store.dispatch(addPerson({ given: 'Alice', surname: 'Brown' }));
  const childId = store.getState().persons[0].id;
  store.dispatch(addParent(childId, { given: 'Carol', surname: 'Brown' }));
  const parent = store.getState().persons.find(p => p.names[0].given ===
'Carol');
  expect(parent).toBeDefined();
```

```
  const rel = store.getState().relationships.find(r =>
r.persons.includes(childId) && r.persons.includes(parent.id));
  expect(rel.type).toBe('biologicalParentChild');
});
```

DevEx, CI/CD & Operations

Repository Structure

```
├── src/
│   ├── components/              # React/Svelte components (TreeCanvas,
NodeCard, etc.)
│   ├── store/                   # State management (slices, reducers, actions)
│   ├── models/                 # Type definitions, validation functions
│   ├── services/               # Storage (IndexedDB), export/import, API
clients
│   ├── i18n/                   # Translation files (en.json, es.json, pt.json)
│   └── pages/                  # Top-level pages (Home, Editor, Settings)
├── public/                     # Static assets (fonts, icons)
├── tests/                      # Unit, integration, E2E tests
├── schemas/                    # JSON Schema definitions
├── .github/workflows/          # CI scripts
└── README.md
```

Code Style & Tools

1. Use **TypeScript** everywhere.
2. Enforce code style with **ESLint** (Airbnb/Google style) and **Prettier**.
3. Use **Husky** and **lint-staged** to run linters/tests on pre-commit.
4. Adopt **Conventional Commits** for commit messages; generate changelogs with `standard-version`.

CI/CD Pipeline Example (GitHub Actions)

1. **Install & Lint:** Install dependencies, run ESLint, and type checks.
2. **Test:** Run unit and integration tests; run E2E tests on headless browsers.
3. **Build:** Build the production bundle using `vite build` or `svelte-kit build`.
4. **Schema Verification:** Run JSON Schema tests against example exports.
5. **Accessibility Audit:** Run axe and lighthouse; fail if issues exceed threshold.
6. **Bundle Size Check:** Fail if compiled JS/CSS exceeds specified budgets.
7. **Deploy:** For SPA track, deploy to Netlify/Vercel; for cloud variant, build Docker image or deploy serverless functions. Use environment variables for secrets.

Release & Versioning

Use **SemVer**: major versions for breaking schema/API changes, minor for new features, patch for bug fixes. Tag releases in Git; publish release notes with features and migration instructions. Maintain a migration script for updating local data when schema version changes.

Feature Flags

Implement a feature flag system to toggle post-MVP functionality (e.g., GEDCOM export, cloud sync). Flags can be stored in local storage or served by the API. Use them to stage features gradually.

Telemetry & Analytics

To measure success metrics, instrument the application with privacy-respecting analytics (e.g., self-hosted Plausible). Collect only aggregate usage data (number of nodes, export events) without storing personal names or birth dates. Provide an opt-in prompt on first use; store consent status locally and sync with the cloud if available. Do not track user data by default (GDPR compliant).

Roadmap & Backlog

MVP Checklist (Estimated 8–10 weeks)

1. **Project scaffolding & design system** (1 wk): set up repository, design tokens, base components.
2. **Core data model & state management** (1.5 wk): implement persons, relationships, events; persistence in IndexedDB.
3. **Canvas & layout engine** (2 wk): implement TreeCanvas with drag-and-drop, keyboard navigation, zoom/pan, auto-layout.
4. **Person and relationship forms** (1 wk): build detail panels with approximate date picker.
5. **Undo/Redo & history** (0.5 wk): implement command stack and UI.
6. **Export/Import (JSON/CSV)** (1 wk): implement serialization, file download/upload, validation.
7. **Accessibility & i18n** (1 wk): keyboard support, screen reader labels, translations for en/es/pt.
8. **Testing & QA** (0.5 wk): write unit/integration tests, run accessibility audits.
9. **Beta & Feedback** (0.5 wk): release to selected users; gather feedback; fix critical issues.

Post-MVP Enhancements

Feature	Description
Cloud Sync & Collaboration	Implement user accounts, remote storage, multi-device access, conflict resolution.
GEDCOM Export/Import	Support export to GEDCOM 5.5.5 and import from GEDCOM with mapping limitations.
Media Attachments	Allow uploading and attaching photos or documents to persons/events.

Feature	Description
Printing & PDF Export	Generate high-quality PDF or SVG exports of the tree with configurable layouts.
Real-Time Collaboration	Enable simultaneous editing by multiple users with WebRTC or CRDT.
AI Assistance	Suggest duplicates, approximate dates, or relationship patterns using machine learning.
Mobile App	Package the SPA as a PWA or native wrapper for offline mobile use.

Risks & Mitigations

Risk	Mitigation
Performance degradation with large trees	Use virtualization and Canvas/WebGL rendering; allow grouping/hiding branches.
Complexity of genealogical relationships	Provide flexible relationship types and custom labels; involve domain experts for validation.
Data privacy concerns	Default to local storage; encrypt data for cloud sync; transparent privacy policy.
Internationalization complexity	Start with en/es/pt; use translation management tools; gather user feedback for edge cases (e.g., double surnames).
Compatibility with GEDCOM	Provide best-effort mapping; warn users about limitations; encourage using JSON for fidelity.
Resource constraints	Plan incremental delivery; use open-source libraries; maintain modular architecture.

Appendices

Glossary

Term	Definition
Genealogy	The study and tracing of lines of family descent.
Person	An individual in the family tree, with name(s), sex/gender, events and relationships.
Proband	The person from whom the genealogy is initiated; often the root of the tree.
Event	A life occurrence (birth, death, marriage, adoption, etc.) with date and place.
GEDCOM	“Genealogical Data Communication” – a plain text file format used for genealogical data exchange [6].

Term	Definition
IndexedDB	A browser API for storing significant amounts of structured data client-side.
CRDT	Conflict-free Replicated Data Type – a data structure that enables concurrent editing without conflicts.

Sample Dataset

To help developers test the application, include a small dataset representing a blended family with adoption and step relationships.

```
{
  "schemaVersion": "1.0",
  "projectId": "demo123",
  "projectName": "Sample Blended Family",
  "persons": [
    {"id": "pA", "names": [{"given": "Alice", "surname": "Garcia", "preferred":
true}], "sex": "female", "gender": "she/her", "events": ["eA"], "notes": [],
"sources": [], "media": []},
    {"id": "pB", "names": [{"given": "Bob", "surname": "Garcia", "preferred":
true}], "sex": "male", "gender": "he/him", "events": ["eB"], "notes": [],
"sources": [], "media": []},
    {"id": "pC", "names": [{"given": "Chloe", "surname": "Garcia", "preferred":
true}], "sex": "female", "gender": "she/her", "events": ["eC"], "notes": [],
"sources": [], "media": []},
    {"id": "pD", "names": [{"given": "Dylan", "surname": "Garcia", "preferred":
true}], "sex": "male", "gender": "he/him", "events": ["eD"], "notes": [],
"sources": [], "media": []},
    {"id": "pE", "names": [{"given": "Elena", "surname": "Smith", "preferred":
true}], "sex": "female", "gender": "she/her", "events": ["eE"], "notes": [],
"sources": [], "media": []}
  ],
  "relationships": [
    {"id": "rM", "type": "marriage", "persons": ["pA", "pB"], "startEventId":
"eM", "endEventId": null, "notes": [], "sources": [], "media": []},
    {"id": "rParent1", "type": "biologicalParentChild", "persons": ["pA",
"pC"], "startEventId": null, "endEventId": null},
    {"id": "rParent2", "type": "biologicalParentChild", "persons": ["pB",
"pC"], "startEventId": null, "endEventId": null},
    {"id": "rParent3", "type": "biologicalParentChild", "persons": ["pA",
"pD"], "startEventId": null, "endEventId": null},
    {"id": "rParent4", "type": "biologicalParentChild", "persons": ["pB",
"pD"], "startEventId": null, "endEventId": null},
    {"id": "rAdopt", "type": "adoptiveParentChild", "persons": ["pE", "pC"],
"startEventId": "eAdopt", "endEventId": null}
  ],
```

```
  "events": [
    {"id": "eA", "type": "birth", "date": {"type": "exact", "value":
"1980-01-02"}, "place": "City X", "description": "Birth of Alice"},
    {"id": "eB", "type": "birth", "date": {"type": "exact", "value":
"1978-03-04"}, "place": "City Y", "description": "Birth of Bob"},
    {"id": "eC", "type": "birth", "date": {"type": "exact", "value":
"2005-05-10"}, "place": "City Z", "description": "Birth of Chloe"},
    {"id": "eD", "type": "birth", "date": {"type": "exact", "value":
"2007-09-15"}, "place": "City Z", "description": "Birth of Dylan"},
    {"id": "eE", "type": "birth", "date": {"type": "exact", "value":
"2010-07-20"}, "place": "City W", "description": "Birth of Elena"},
    {"id": "eM", "type": "marriage", "date": {"type": "exact", "value":
"2003-06-15"}, "place": "Town Hall", "description": "Marriage of Alice and
Bob"},
    {"id": "eAdopt", "type": "adoption", "date": {"type": "exact", "value":
"2013-04-01"}, "place": "Court", "description": "Adoption of Chloe by Elena"}
  ],
  "sources": [],
  "media": []
}
```

FAQ

1. **Where is my data stored?** – In the MVP, all data is stored locally in your browser's IndexedDB and never leaves your device unless you choose to export or enable cloud sync. When cloud sync is enabled post-MVP, data is encrypted and stored on our servers.
2. **Do I need to create an account?** – No account is required for offline use. You can optionally create an account to enable backups and multi-device sync once that feature is released.
3. **Can I add same-sex partners?** – Yes. Relationships are modeled symmetrically; there is no assumption of gender. GEDCOM uses `HUSB` / `WIFE` tags but explicitly states they should not imply gender [4].
4. **How do I represent step-parents or adopted children?** – When adding a parent, select the relationship type (biological, adoptive, step) in the relationship manager. Step relationships can connect a parent to a child without marking them as biological. This mirrors guidelines from genealogical tools [5].
5. **What happens if I import data and there are duplicates?** – The import process identifies potential duplicates using names and dates. You will be prompted to merge or keep separate records.
6. **Is GEDCOM export fully compatible?** – We provide best-effort mapping to GEDCOM 5.5.5, but the format has limitations (binary gender, limited support for step/adoptive relationships). Some information may be stored in notes or custom tags.
7. **What if I close the browser accidentally?** – Your data is autosaved to IndexedDB. When you return to the site, you can open your previous projects. It is still recommended to export backups.

Open Questions & Assumptions

Question	Proposed Default	Note
Should users be able to specify custom relationship types?	Yes, allow a “custom” relationship type with a user-defined label.	Enhances flexibility but may limit export compatibility.
How will we handle names in cultures with patronymic/ matronymic structures?	Support multiple surname fields and allow custom order.	Further research needed for specific cultures.
Will GEDCOM v7 be supported?	Not in MVP; evaluate adoption of GEDCOM 7, which uses UTF-8 and GEDZip [8] .	Many tools still use 5.5.5; compatibility may vary.
Do we enforce two parents per child?	Allow more guardians but highlight when exceeding two biological/ adoptive parents.	Step and guardian relationships are separate types.

Self-Check List

To ensure completeness, review the following checklist. Each item should be addressed in the document above.

1. **PRD with personas, user stories, acceptance criteria** – done.
2. **UX flows + wireframes (ASCII) + accessibility & i18n plan** – done.
3. **Data model + ERD + JSON Schema + GEDCOM mapping notes** – done.
4. **Export/Import spec with example files and versioning** – done.
5. **Architecture for local-first and cloud-enabled options + diagrams** – done.
6. **API (OpenAPI 3.1) stub** – provided for cloud variant.
7. **FE component spec with props/events + state/undo strategy** – done.
8. **Validation rules + testing strategy + example tests** – done.
9. **DevEx + CI/CD + release/versioning + telemetry plan** – done.
10. **Roadmap/backlog with estimates, risks** – done.
11. **Appendices: glossary, sample dataset, FAQ** – done.
12. **Explicit coverage of all edge cases (multiple parents/guardians, adoption, step/half siblings, multiple marriages/partners, non-binary and custom genders, unknown parents, uncertain dates, duplicate detection, large trees, cycles, international name formats, diacritics, RTL languages, mobile users, offline edits with later sync conflicts, printing, exporting incomplete trees)** – integrated throughout document.

---

[1] Using About, Between, After and Before Dates In Genealogy - https://mexicangenealogy.com/using-about-between-after-and-before-dates-in-genealogy/

2 Understanding Success Criterion 2.4.3: Focus Order | WAI | W3C
https://www.w3.org/WAI/WCAG22/Understanding/focus-order

3 WCAG 2.2 AA | Summary and Web Accessibility Checklist
https://www.levelaccess.com/blog/wcag-2-2-aa-summary-and-checklist-for-website-owners/

4 8 9 GEDCOM - Wikipedia
https://en.wikipedia.org/wiki/GEDCOM

5 Mapping Modern Roots: A Guide to Nontraditional Family Trees
https://familytreemagazine.com/strategies/guide-to-nontraditional-family-trees/

6 7 GEDCOM
https://www.gedcom.org/

10 Cryptographic Storage - OWASP Cheat Sheet Series
https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

11 Transport Layer Security - OWASP Cheat Sheet Series
https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html