The Vault: Project Overview and Technical Architecture (2025)

A fully automated, local-first cataloging system for hundreds of archival music concert recordings, combining structured metadata, rich media screenshots, and a Netflix-style browsing experience.

This document replaces all previous versions of the project summary.

---

1. Project Purpose

The Vault aims to create a single, authoritative source of truth for a large private collection of rare concert recordings. Each show includes:

- Setlists
- Venue, city, country
- Taper lineage and recording source info
- Technical metadata (file formats, disc structures, VOB/TS groups)
- High-quality screenshots (3–4 per show)
- Master hard-drive reference

The project delivers:

- A cleaned, structured **master CSV** representing the entire collection.
- A unified **shows.json** derived from the CSV.
- A **static website** that feels like Netflix: dark mode, polished, fast.
- A local development workflow that consumes *real* metadata and screenshots.
- A highly automated system requiring minimal manual effort.

No online hosting is currently used; everything runs locally for privacy, speed, and simplicity.

---

2. High-Level Workflow (2025)

1. Data Collection & Metadata Processing

- Thousands of archival concert folders across multiple external drives.
- Automated Python scripts read each folder:
- Extract show metadata from TXT/DOC files.
- Parse setlists heuristically.
- Detect multiple VOB/TS files and group them.
- Capture lineage, taper, venue info.
- Output is written into a **master CSV**.

2. Screenshot Capture

Fully automated screenshot generation (VLC or mpv): - 3–4 full-resolution images at predictable timestamps. - Written to `public/images/<show-id>/...`. - Browser scales these for thumbnails.

3. CSV → shows.json

- Python conversion pipeline cleans, normalises, and outputs:
- Artist
- Date
- Venue, city, country
- Setlist
- Source/lineage
- Screenshot paths
- Drive location

4. Figma Make → Real UI Code

Figma Make produces a ZIP containing:

```
(zip root)/
  index.html
  package.json
  vite.config.ts
  tsconfig.json
  tsconfig.node.json
  src/
  public/   (Figma's placeholder data – ignored/merged)
```

This code is **not fully accurate** to the real data structure in The Vault. Therefore, we run a custom automation script.

5. Automated Sync & Build Pipeline

The core of the system.

A shell script `update_from_figma.sh`: - Detects latest ZIP in `~/Downloads`. - Extracts the ZIP. - Treats the ZIP's **src folder** as the true project root. - Syncs all updated UI code into the stable project. - **Preserves real data:** - `public/images/` - `public/shows.json` - Installs dependencies. - Ensures Tailwind CSS directives are injected. - Runs a full `npm run build`.

6. Live Preview via Vite

Local dev server:

```
npm run dev
```

Visit:

```
http://localhost:5173/
```

This shows **real** concert data and **real** screenshots integrated into the Netflix-style UI.

---

3. The Stable Local Project Structure

Final folder layout after syncing:

```
the-vault/
  index.html
  package.json
  vite.config.ts
  tsconfig.json
  tsconfig.node.json

  public/
    images/           # All real screenshots (preserved)
    shows.json        # Real metadata JSON

  src/
    App.tsx
    main.tsx
    components/
    styles/
    index.css

  update_from_figma.sh
  node_modules/
```

This structure is required for: - Vite dev server - Tailwind/PostCSS pipeline - Figma Make compatibility

---

4. Figma Make → Local Sync Script

`update_from_figma.sh` performs the following:

Automated ZIP Handling

- Locates newest `the-vault*.zip` in `~/Downloads` automatically.
- Extracts ZIP to a temporary directory.
- Identifies the correct project root (`zip/src`).

Code Syncing

- Syncs UI code into the local project root.
- Syncs Figma `public/` while *preserving* existing images & JSON.

Dependency Management

- Runs `npm install` as needed.
- Ensures Tailwind CSS dependencies are installed.
- Injects Tailwind directives into `index.css` if missing.

Build Automation

- Always runs `npm run build` at the end.

This script eliminates all manual copying, cleaning, reinstalling, or rebuilding.

---

5. One-Command Workflow (Aliases)

To avoid memorising commands, the workflow is wrapped inside `zsh` functions. These live in `~/.zshrc`.

vault-go

```
vault-go() {
  cd /Users/ko/Desktop/Projects/the-vault || return
  ./update_from_figma.sh
  # open http://localhost:5173/     # optional
  npm run dev
}
```

Run this after every Figma export:

```
vault-go
```

vault-build (optional)

```
vault-build() {
  cd /Users/ko/Desktop/Projects/the-vault || return
  ./update_from_figma.sh
}
```

Useful when changing **only** metadata (e.g. new shows.json).

vault-help

```
vault-help() {
  echo ""
  echo "Vault Commands:"
  echo "  vault-go      → Update project from Figma ZIP + run dev server"
  echo "  vault-build   → Update project from Figma ZIP only"
  echo ""
  echo "Other helpful commands:"
  echo "  alias         → Show all aliases"
  echo "  functions     → Show all functions"
  echo ""
}
```

When you forget commands, simply type:

```
vault-help
```

---

6. Why Local Development (No GitHub/Netlify)

The Vault is currently a **private collection** and must handle: - Thousands of screenshots - Several gigabytes of media metadata - Frequent iteration - Zero exposure of private archives or personal collection

Local development provides:

✔️ Privacy

No hosting or uploads of private material.

✔️ Speed

Vite dev server is near-instant.

✔️ Simplicity

One command updates everything.

✔️ Control

The project works offline and without platform limitations.

Future hosting (optional) could be: - GitHub Pages - Netlify - Cloudflare Pages - S3 bucket

But not needed at this stage.

7. Developer Checklist

After making changes in Figma Make:

1. Export latest ZIP to `~/Downloads`.
2. Run:

```
vault-go
```

3. Browser preview opens at:

```
http://localhost:5173/
```

After updating metadata (Master CSV → shows.json):

1. Regenerate `public/shows.json`.
2. Run:

```
vault-build
```

After adding new screenshots:

1. Add to `public/images/<show-id>/...`.
2. Refresh browser.

If Terminal commands are forgotten:

- `vault-help`
- `alias`
- `functions`

---

8. Future Enhancements

- Automated CSV → shows.json converter (button-triggered)
- Automated VLC/mpv screenshot pipeline
- Automated artist-grouping logic
- Full-text search improvements
- Optional deployment to a private remote server
- Adding nightly backups of JSON + images
- UI improvements in Figma Make driven by real dataset

---

9. Summary

The Vault is now a: - Fully local - Fully automated - Netflix-style - Metadata-driven - Private archival system

One command (`vault-go`) updates, rebuilds, and launches the entire interactive UI using real show metadata and high-quality screenshots.

This document represents the current authoritative technical specification of the system.