NAS (Network Accessed Storage) — Comprehensive Documentation

A self-hosted, password-protected web app to browse, preview, and stream files from your local or mounted drives.

- Backend: FastAPI (Python)
- Frontend: Jinja templates + vanilla JavaScript
- Auth: Signed session cookies (itsdangerous)
- Media: Image thumbnails, video posters, HTTP range streaming
- Previews: Rich in-popup viewers for many document and text formats
- Platform: Linux, macOS, Windows

This document covers what the project does, how it's structured, how each major part works, all dependencies, and detailed setup for development, production, and remote access (including Tailscale and Windows Funnel).

---

1) Overview

NAS provides a secure web interface to browse and manage files on one or more local/mounted drives. It emphasizes: - Simplicity: No databases, no heavy frameworks on the frontend. - Safety: Authenticated access, sanitized path handling. - Speed: Cached thumbnails/posters, HTTP range streaming, lazy image loading. - UX: Clean responsive UI, list/grid views, and in-popup previews.

Common use cases - Self-hosted NAS browser on a home server or workstation - Quickly preview and download files remotely (e.g., via Tailscale) - Lightweight "media shelf" with thumbnails and streaming for supported codecs

---

2) Features

- Authentication
  - Login via username/password
  - Signed session cookies
- Drive discovery and browsing
  - Lists allowed roots and their directories
  - Safe path join to block traversal
- File operations
  - Download
  - Upload (toggle via config)
  - Delete (toggle via config; optional recursive dir delete)
- Previews (in modal popups)

  - Images: JPG/JPEG, PNG, WEBP, GIF, BMP, HEIC/HEIF, AVIF, TIFF
  - Video: MP4, WEBM, MOV, M4V, AVI, MKV, OGV/OGG
    - Playback depends on browser codecs (MKV support is UI-level; decoding must be supported by the browser)
  - PDF: Embedded inline (iframe)
  - Docs: DOCX (Mammoth.js), XLSX/XLS/CSV (SheetJS)
  - Text/Code: JSON/JSO/IPYNB (pretty JSON), XML (pretty indentation), YAML/YML (pretty using js-yaml if available), TXT, CPP, PY
  - Legacy DOC/PPT: no in-browser preview (use Download)
- Thumbnails and posters
  - Images via Pillow (auto orientation from EXIF)
  - HEIC/HEIF via pillow-heif (optional)
  - Video posters via ffmpeg
- Search
  - Server-side recursive name search with depth/limit options
- UI
  - List/grid view toggle
  - Lazy thumbnail loading with deterministic ordering
  - Preview/Download/Delete actions in the table
  - In-popup viewers (no server plugins required)

Note about the actions row in the UI - The “Open” link has been removed and replaced by a “Download” link adjacent to “Preview” and “Delete” to match the current UX.

---

3) Architecture and Flow

High-level - FastAPI application exposes HTML pages and a simple authenticated API. - Jinja templates render pages; JavaScript enhances UI and calls API endpoints. - Authentication middleware validates sessions; unauthenticated users are redirected to the login page. - File operations are performed with careful path validation against allowed roots.

Request flow 1. User logs in → server sets a signed cookie with session info. 2. User visits “Browse” → FastAPI renders template (includes app.js). 3. app.js calls `/api/drives` and `/api/list` to fill the UI. 4. User actions (Preview/Download/Delete/Upload/Search) call the respective API endpoints. 5. For previews, the frontend decides which modal to open (image/video/text/pdf/spreadsheet/docx) and fetches the file or a processed representation (e.g., render_image, thumb, stream).

---

4) Repository Layout

```
NAS/
├─ hdd_browser/
│  ├─ app/
│  │  ├─ main.py               # FastAPI app entry (mounts routes, templates, static)
│  │  ├─ auth.py               # Authentication, session cookie handling
│  │  ├─ config.py             # Loads .env / environment variables
│  │  ├─ security.py           # Public path rules, safe path join, guards
│  │  ├─ drive_discovery.py    # Drive/mount enumeration, allowed roots logic
│  │  ├─ file_ops.py           # Read/list/download/stream/search/upload/delete
│  │  ├─ thumbnailer.py        # Image/video thumbnail rendering & caching
│  │  ├─ heic_init.py          # Pillow HEIC/HEIF init (optional)
│  │  ├─ templates/            # Jinja2 templates (home, browse, search, login)
│  │  └─ static/
│  │     ├─ css/               # Styles
│  │     └─ js/              # Entire frontend behavior (UI + client viewers)
│  └─ requirements.txt         # Python dependencies
└─ README.md                   # Project summary (shorter than this doc)
```

Notes - The exact template names may vary; look inside `hdd_browser/app/templates/`. - Static assets (CSS/JS) are served from `hdd_browser/app/static/`.

---

5) Backend (FastAPI) Components

The following describes typical responsibilities of the backend modules. Names are based on the project layout shown above.

- main.py
  - Creates the FastAPI app.
  - Configures Jinja2Templates, static routes, and auth middleware.
  - Defines page routes:
    - GET /login, POST /auth/login, POST /auth/logout
    - GET / (home), GET /browse, GET /search
  - Defines or includes API routes (see API Reference).
- auth.py
  - Validates username/password (from environment or config).
  - Sets/clears signed session cookie (via itsdangerous).
  - Provides dependencies to enforce authentication on routes.
  - Public paths (login, static) bypass auth; others require valid session.
- config.py
  - Reads environment variables and optional .env file.
  - Exposes settings (APP_NAME, DEBUG, ENABLE_UPLOAD/DELETE/THUMBNAILS, THUMB_CACHE_DIR, FFMPEG_PATH, ALLOWED_ROOTS, etc.).
- security.py
  - Holds allowlist/denylist of public paths (e.g., /auth/login, /static/*).
  - Safe path joining utilities (avoid “..” path traversal).
  - Guards API endpoints (ensures requests are authenticated and paths are within allowed roots).

- drive_discovery.py
  - Enumerates drives/mount points on the host OS.
  - Applies ALLOWED_ROOTS if set; otherwise discovers common mounts.
  - Returns stable IDs/names for frontend selection.
- file_ops.py
  - Directory listing with metadata (name, size, modified, mime, is_dir).
  - File download (FileResponse).
  - HTTP range streaming for audio/video.
  - Upload support (multipart form).
  - Delete support (file or recursive directory).
  - Text preview endpoint (reads only up to a safe limit).
  - Image rendering endpoint (e.g., downscale to fit UI).
  - Thumbnail endpoint (images/videos).
  - Search endpoint (recursive name search with depth/limit).
- thumbnailer.py
  - Generates and caches thumbnails:
    - Images: Pillow (auto orientation, resize)
    - Videos: ffmpeg captures a frame for poster/thumbnail
  - Cache location controlled with THUMB_CACHE_DIR.
- heic_init.py
  - Optional import/init for pillow-heif to enable HEIC/HEIF decoding.
  - If present and configured, images in HEIC are converted to JPEG for preview/thumbnail.

---

6) Frontend (app/static/js/app.js) Components

The entire UI behavior is implemented in a single file (`app.js`). Below is a guided map of the major functions and how they interact.

Core helpers - fetchJSON(url): Fetches JSON with credentials and no cache; throws on non-OK responses. - formatSize(bytes): Human-readable file sizes. - escapeHtml(s): Safe HTML text rendering. - absoluteHref(pathOrUrl): Normalizes to an absolute URL (used for plugins).

Multi-CDN loader + optional vendor fallback - loadScriptFromAny(urls, testFn, {optional}): Attempts to load a script from multiple URLs; verifies via `testFn` and can be optional. - loadCssFromAny(urls): Injects the first available stylesheet from a set of URLs. - vendorUrl(path): If `window.PPTX_VENDOR_BASE` is set (e.g., "/static/vendor/…"), local copies will be preferred for offline/air-gapped setups.

Client-side viewer libraries (lazy-loaded) - ensureMammoth(): Loads Mammoth.js to render DOCX → HTML. - ensureXLSX(): Loads SheetJS to render XLSX/XLS/CSV → HTML tables. - ensurePPTXViewer(): Loads pptxjs and dependencies (jQuery, JSZip, d3, nvd3). Note: PPT/PPTX previews are not advertised in the README; this viewer is experimental in the codebase and may be disabled/not surfaced in production UI. - ensureJSYAML(): Loads js-yaml for pretty YAML formatting (optional).

Pretty printers - prettyXml(xml): Collapses and re-indents XML content for display.

Thumbnails: deterministic and lazy - ThumbQueue: Serializes image loading (default concurrency = 1) to keep ordering stable. You can increase via `window.THUMB_MAX_CONC`. - IntersectionObserver: Lazily starts loading thumbnails when they come into view.

Media detection - isPreviewableImage(mime, name): Decides whether to use the image preview path. - isPreviewableVideo(mime, name): Accepts MP4, WEBM, MOV, M4V, AVI, MKV, OGV/OGG or `video/*` mime (playback depends on browser codecs). - isMedia: union of the above.

Modal components - createMediaModal: For images and videos (with optional poster thumbnails). Includes Download and (Prev/Next) controls. - createTextModal: For text/code (JSON, IPYNB, XML, YAML, TXT, CPP, PY, etc.). Includes a Copy button and Download link. - createHtmlModal: For rich HTML (DOCX, XLSX/CSV render results). Includes Download; the "Open" action has been removed per current UX. - createPdfModal: Embeds PDFs in an iframe; includes Download.

Client viewers - openDocxInModal({title, downloadHref}): Fetches file blob → Mammoth → HTML → open HtmlModal. - openSpreadsheetInModal({title, downloadHref, ext}): Fetches file blob → SheetJS → HTML tables per sheet → open HtmlModal. - openPptxInModal({title, downloadHref}): Experimental pptxjs rendering (not advertised in README); honors fullscreen only if `screenfull` is available.

Browse page lifecycle - initBrowser(): - Loads drives (`/api/drives`), lists directory (`/api/list`), wires Upload, Sort, View Toggle, and "Up" navigation. - Renders either a table view (renderTable) or a grid view (renderGrid). - For table rows, Actions include: - Preview (opens the appropriate modal) - Download (direct link with `download` attribute) - Delete (if enabled) - The old "Open" action has been removed and replaced with "Download" next to "Preview" and "Delete."

- openNonMediaPreview(name):
  - Calls `/api/preview` for metadata/snippets.
  - Decides modal type:
    - Text/code → pretty prints JSON, XML, YAML where possible.
    - DOCX/XLSX/CSV → opens client viewer modals.
    - PDF → in iframe.
    - Legacy DOC/PPT → shows a note to use Download.
    - Otherwise → falls back to the inline preview panel at the bottom.
- renderTable / renderGrid:
  - Build list/grid entries.
  - Lazy load thumbnails for images via `/api/thumb` and videos for posters when opening media modals.

Search page - initSearch(): - Calls `/api/search` with query and displays results list with sizes.

---

7) API Reference

All non-public endpoints require a valid session cookie. Login via the web UI at `/login`.

Auth - GET /auth/login (page): Login form. - POST /auth/login: Body form fields `username`, `password`. On success, sets a signed session cookie and redirects. - POST /auth/logout: Clears session.

Drives and listing - GET /api/drives - Response: JSON array of drive objects or strings (id/label pairs). - GET /api/list?drive_id=…&rel_path=… - Returns current path + entries. - Entry fields: name, is_dir, mime (best effort), size (bytes), modified (epoch seconds).

Previews and media - GET /api/preview?drive_id=…&rel_path=… - For text/code formats returns {name, mime, text, truncated}. - For binary formats returns metadata only (frontend decides what to do). - GET /api/download?drive_id=…&rel_path=… - Direct file download (FileResponse). - GET /api/stream?drive_id=…&rel_path=… - Range streaming endpoint for audio/video. - GET

/api/thumb?drive_id=...&rel_path=...&size=180 - Thumbnail for images/videos (size = max dimension). - GET /api/render_image?drive_id=...&rel_path=...&max_dim=1600 - Returns downscaled image to fit modal.

Search - GET /api/search?drive_id=...&q=...&max_depth=...&limit=... - Simple recursive name search; returns list of path/size pairs.

Upload and delete (if enabled) - POST /api/upload (multipart/form-data) - Fields: drive_id, rel_path, file (file field repeated for multiple files) - POST /api/delete - Form fields: drive_id, rel_path, optional recursive=1 for directories.

Example: curl a download

```
curl -v -b "session=YOUR_COOKIE" \
  "http://localhost:8080/api/download?drive_id=C%3A&rel_path=Users%2Fme%2Ffile.pdf" \
  -o file.pdf
```

---

8) Dependencies and Libraries

Python (requirements.txt) - fastapi: web framework - uvicorn[standard]: ASGI server - itsdangerous: cookie signing - psutil: drive/mount discovery - jinja2: server-side templates - python-multipart: file uploads - pydantic: request/response models (FastAPI) - Pillow: image processing - pillow-heif: HEIC/HEIF decoding (optional)

External JS libraries (lazy-loaded in the browser) - Mammoth.js: DOCX → HTML (client-side) - SheetJS (XLSX): XLSX/XLS/CSV → HTML tables (client-side) - js-yaml: YAML parsing and pretty printing (optional) - pptxjs (+ jQuery, JSZip, d3, nvd3, screenfull): Experimental PPTX renderer (the project README does not advertise PPT/PPTX preview; this is for developers and may be disabled in production)

Browser APIs/Techniques - IntersectionObserver: lazy load thumbnails offscreen - Clipboard API for copy in text modal - Deterministic image loading queue for stable visual order

---

9) Installation and Setup

9.1 Local Development (Linux/macOS/Windows)

1) Clone

```
git clone https://github.com/Atharva0177/NAS.git
cd NAS
```

2) Python environment

```
python3 -m venv .venv
# Windows PowerShell: .\.venv\Scripts\Activate.ps1
source .venv/bin/activate
pip install -r hdd_browser/requirements.txt
```

3) Configure environment

- Minimal variables:
  - AUTH_USERNAME
  - AUTH_PASSWORD
  - SESSION_SECRET (use a long random string)

- Optional: ENABLE_UPLOAD, ENABLE_DELETE, ENABLE_THUMBNAILS, THUMB_CACHE_DIR, ALLOWED_ROOTS, FFMPEG_PATH

Example (`.env`)

```
APP_NAME=NAS
DEBUG=True
AUTH_USERNAME=admin
AUTH_PASSWORD=secret
SESSION_SECRET=change_me_long_random
ENABLE_UPLOAD=True
ENABLE_DELETE=False
ENABLE_THUMBNAILS=True
THUMB_CACHE_DIR=.thumb_cache
ALLOWED_ROOTS=/mnt/storage,/home
```

4) Run the server

```
uvicorn hdd_browser.app.main:app --host 0.0.0.0 --port 8080 --reload
```

5) Open

```
http://localhost:8080
```

9.2 Production (Uvicorn/Systemd on Linux)

Start manually:

```
uvicorn hdd_browser.app.main:app --host 0.0.0.0 --port 8080
```

Systemd service example:

```
[Unit]
Description=NAS (FastAPI)
After=network.target

[Service]
Type=simple
User=YOUR_USER
WorkingDirectory=/path/to/NAS
Environment=PATH=/path/to/NAS/.venv/bin
ExecStart=/path/to/NAS/.venv/bin/uvicorn hdd_browser.app.main:app --host 0.0.0.0 --port
8080
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Enable and start:

```
sudo systemctl enable nas
sudo systemctl start nas
sudo systemctl status nas
```

9.3 Windows Service (Optional with NSSM)

1) Install NSSM
2) Elevated PowerShell:

```
nssm install NAS
# NSSM GUI:
# Application: C:\Path\To\NAS\.venv\Scripts\uvicorn.exe
# Arguments:   hdd_browser.app.main:app --host 127.0.0.1 --port 8080
# Startup dir: C:\Path\To\NAS
```

3) Set auth/config environment variables via System → Advanced → Environment Variables or NSSM "Environment".
4) Start:

```
nssm start NAS
```

9.4 Remote Access with Tailscale

Tailscale provides a private, encrypted network between devices.

Install - Download Tailscale on your NAS and client devices. - Sign in with the same account. - On the NAS (Windows example):

```
& "C:\Program Files\Tailscale\tailscale.exe" status
& "C:\Program Files\Tailscale\tailscale.exe" ip -4
```

Run NAS bound to all interfaces:

```
uvicorn hdd_browser.app.main:app --host 0.0.0.0 --port 8080
```

Access from another tailnet device: - Without MagicDNS: `http://<tailscale-ip>:8080` - With MagicDNS (recommended): `http://<device-name>.<tailnet>.ts.net:8080`

Security for remote use: - `DEBUG=False` - Strong `AUTH_USERNAME`, `AUTH_PASSWORD`, very long `SESSION_SECRET` - Consider `ENABLE_DELETE=False`

9.5 Public HTTPS via Tailscale Funnel (Windows)

Funnel publishes your local service publicly under an HTTPS `.ts.net` domain.

Prerequisites - Tailscale v1.54+ on Windows - Funnel enabled in Admin Console (Settings → Funnel) - MagicDNS enabled (Admin Console → DNS) - Device allowed to obtain HTTPS certs (Let's Encrypt)

Steps

```
# 1) Start NAS locally
.\.venv\Scripts\Activate.ps1
uvicorn hdd_browser.app.main:app --host 127.0.0.1 --port 8080

# 2) Serve HTTPS from device's .ts.net name to local NAS
& "C:\Program Files\Tailscale\tailscale.exe" serve https / http://127.0.0.1:8080

# 3) Enable public Funnel on port 443
& "C:\Program Files\Tailscale\tailscale.exe" funnel 443 on

# 4) Check status
& "C:\Program Files\Tailscale\tailscale.exe" serve status
```

You'll get a URL like:

```
https://<device>.<tailnet>.ts.net/
```

Disable later:

```
& "C:\Program Files\Tailscale\tailscale.exe" funnel 443 off
& "C:\Program Files\Tailscale\tailscale.exe" serve reset
```

Custom domains - Funnel uses `.ts.net`. For a custom domain, deploy a standard reverse proxy (nginx/traefik) or a different tunneling solution that supports custom DNS.

---

10) Configuration

Environment variables - `APP_NAME`: Shown in UI header - `HOST`, `PORT`: Bind address - `DEBUG`: True/False (don't use True on the public Internet) - `AUTH_USERNAME`, `AUTH_PASSWORD`: Credentials for login - `SESSION_SECRET`: 16+ character secret (long random recommended) - `ENABLE_UPLOAD`: True/False - `ENABLE_DELETE`: True/False (dangerous; consider False for remote) - `ENABLE_THUMBNAILS`: True/False - `THUMB_CACHE_DIR`: Folder for cached thumbnails/posters (e.g., `.thumb_cache`) - `FFMPEG_PATH`: Optional explicit ffmpeg path - `ENABLE_HEIC_CONVERSION`: True/False - `ALLOWED_ROOTS`: Comma-separated root directories allowed for browsing (strongly recommended)

Frontend toggles (via template/global) - `THUMB_MAX_CONC`: Increase to load multiple thumbnails concurrently (default 1 keeps ordering stable).

---

11) Security Notes

- Session cookies are signed; keep `SESSION_SECRET` long and private.
- Always set `ALLOWED_ROOTS` to confine the browser to known paths.
- Avoid enabling `ENABLE_DELETE` for remote access or production unless necessary.
- Use HTTPS whenever exposing across untrusted networks:
  - Tailscale Funnel provides automatic TLS for `.ts.net` domain.

---

12) Performance and Tuning

- Thumbnails
  - Default serial loading (stable ordering). Increase `THUMB_MAX_CONC` if you need faster image grids on powerful clients.
  - Thumbnail cache prevents re-processing on the server.
- Previews
  - Text previews are truncated (~1.5 MB) to keep the UI snappy.
- Video streaming
  - Uses HTTP range responses for fast seeking.
  - For unsupported codecs (e.g., some MKV), the browser may fail to play; the Download link is still available.
- ffmpeg
  - Ensure ffmpeg is installed and on PATH; set `FFMPEG_PATH` if needed.

---

13) Troubleshooting

- Login loop / 401
  - Ensure `SESSION_SECRET` is 16+ characters.
  - Check browser cookie settings.

    - Confirm server time is accurate.
- No thumbnails
    - `ENABLE_THUMBNAILS=True`
    - `ffmpeg` available for video posters
    - Check write permissions for `THUMB_CACHE_DIR`
- HEIC not rendering
    - Install `pillow-heif` (already in requirements)
    - Confirm `heic_init.py` loads without error
- PPT/PPTX previews
    - Not supported in production (use Download).
    - The codebase includes experimental pptxjs wiring for developers; compatibility is limited.
- MKV won't play
    - Browser codec support is required. Use Download if playback fails.

---

14) Extensibility and Roadmap

Ideas for extensions - Per-user auth and roles (beyond single user) - WebDAV or S3 backends - Archive previews (ZIP, RAR) - Syntax highlighting via Prism.js in text modal - Server-side DOC/PPT converters for legacy Office formats - Thumbnail workers (async queue) for very large libraries

Where to add features - Backend endpoints: `file_ops.py` (for new preview routes or converters) - Frontend preview routing: `openNonMediaPreview` in `app/static/js/app.js` - UI elements: corresponding Jinja templates in `templates/`
