FaceForge

A local-first, self-hosted **Asset Management System (AMS)** focused on **entities** (real or fictional, human or non-human) and the **assets** attached to them.

FaceForge is designed to be a stable, integration-friendly “boring core” for storing metadata + files, while advanced features (recognition, graph visualization, training, third-party integrations) live in optional plugins.

Status

This repository is currently **docs-first scaffolding**:

- `core/` and `desktop/` are placeholders today.
- The “what it is / how it should work” is defined in the spec and the MVP sprint plan.

If you’re arriving here to understand the project, start with:

- Design spec: docs/FaceForge Core - Project Design Specification - v0.2.9.html
- Roadmap / implementation sequence: docs/FaceForge Core - MVP Sprint Sequence.html

What FaceForge is (and isn’t)

FaceForge is:

- A desktop-managed local services bundle (no end-user Docker)
- A local HTTP API + web UI for CRUD of entities, assets, relationships, jobs, and plugin configuration
- Built for large local datasets: streaming download + HTTP range/resume support
- Integration-first: external tools should be able to query Core via a stable OpenAPI-documented API

FaceForge is not (in Core):

- Face recognition, embedding generation, LoRA training, graph rendering, or deep third-party integrations
- A cloud-hosted SaaS requirement

Those capabilities are intended to ship as plugins that talk to Core over its public APIs.

High-level architecture (intended)

- **FaceForge Desktop** (Tauri v2): orchestrates local components (Core server, optional sidecars like object storage, plugin runners), manages lifecycle, and opens the UI.
- **FaceForge Core** (planned: Python 3.11/3.12 + FastAPI + Uvicorn): serves a versioned API under `/v1/...`, plus `/docs` and `/redoc`, and hosts the web UI.
- **Storage**: transparent, user-controlled storage paths. Default object storage is intended to be an S3-compatible local endpoint (SeaweedFS), but Core must also work in filesystem-only mode.
- **Metadata DB**: SQLite, using a “relational spine + JSON fields” approach (entities/assets/relationships/jobs + flexible descriptors).

Core conventions (from the spec)

When implementation starts, the repo will follow these conventions:

- Network defaults: bind to `127.0.0.1`; require a per-install token for non-health endpoints.
- API routing: versioned under `/v1/...` with OpenAPI always kept accurate.
- Local-first storage contract: `FACEFORGE_HOME` is the root data directory; Core creates subfolders under it (e.g. `db/`, `logs/`, `config/`, `plugins/`).
- Asset downloads: streaming-first, HTTP range support (resume-friendly), no opaque container volumes.

Repository layout

- `core/` — planned Core API service (currently placeholder)
- `desktop/` — planned Tauri desktop orchestrator (currently placeholder)
- `docs/` — design spec + MVP sprint sequence (current source of truth)
- `brand/` — logos, favicon assets, and fonts used for UI branding

Getting started (today)

There isn’t runnable Core/Desktop code in this repo yet.

To get productive right now:

- Read the Project Design Specification:
  - HTML
  - Markdown
  - PDF
- Follow the MVP sprint sequence to implement in small, shippable increments:
  - HTML
  - Markdown
  - PDF
- For implementers using AI assistance (e.g. GitHub Copilot), follow these guidelines

  - Markdown: .github/copilot-instructions.md

Contributing

Contributions are welcome, especially as the repo moves from docs → scaffolding → MVP.

- If you're proposing changes, align them to the spec and the sprint sequence.
- Prefer small PRs that complete a sprint deliverable or a clearly scoped subtask.

License

See LICENSE.