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** with an initial runnable Core implementation:

- `core/` contains a FastAPI app with versioned routing (`/v1/...`), token auth defaults, SQLite migrations, and initial real endpoints (starting with Entities CRUD).
- `desktop/` contains a Tauri v2 Desktop shell MVP (wizard + tray + local process orchestration).
- 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. Core supports filesystem-only mode and an optional S3-compatible provider (intended: SeaweedFS S3 endpoint) with upload-time routing + automatic fallback.
- **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/`, `assets/`, `logs/`, `config/`, `plugins/`).
- Asset downloads: streaming-first, HTTP range support (resume-friendly), no opaque container volumes.

Repository layout

Quick map of what lives where:

```
  desktop/      # Tauri desktop shell (Sprint 12 MVP)
faceforge/
  core/         # FastAPI Core service (runnable today)
  desktop/      # Tauri desktop shell (placeholder for now)
  docs/         # Project spec + MVP sprint plan (source of truth)
  scripts/      # Dev scripts (PowerShell) to run/check Core using .venv
  brand/        # Logos, favicon, fonts (UI branding assets)
```

Core service code is a standard Python package under:

```
core/
  src/faceforge_core/        # app entrypoint + API + DB + storage
    app.py                   # FastAPI app factory / wiring
    api/v1/                  # versioned HTTP routes
    db/                       # SQLite schema + migrations + queries
    internal/                 # internal CLIs/utilities (dev/admin)
  tests/                      # pytest tests for Core
  pyproject.toml              # Core packaging/deps
```

If you're new and wondering "where do I add a route?", start at:

- `core/src/faceforge_core/api/v1/router.py` (v1 router)
- `core/src/faceforge_core/app.py` (app wiring)

Getting started (today)

FaceForge Core is runnable in dev today, and FaceForge Desktop can orchestrate it.

Prerequisites

Before you start, make sure these tools are installed.

- Rust toolchain (`rustc`, `cargo`)
  - Verify: `rustc --version` and `cargo --version`
  - Install (recommended): Rustup from https://rustup.rs
    - Or via WinGet: `winget install --id Rustlang.Rustup -e`
- Tauri CLI (`cargo tauri`)
  - Verify: `cargo tauri --version`
  - Install: `cargo install tauri-cli`
- Python 3.12.x
  - Verify: `python --version`
  - Install: https://www.python.org/downloads/ (make sure "Add Python to PATH" is enabled)
    - Or via WinGet: `winget install --id Python.Python.3.12 -e`

If you have multiple Pythons installed, ensure `python` resolves to 3.12.x in the same terminal where you run `./scripts/*.ps1`.

Run Desktop (dev)

Prereq: Rust toolchain (stable).

From the repo root:

- `cargo tauri dev --manifest-path .\desktop\src-tauri\Cargo.toml`

What it does:

- First-run wizard to choose `FACEFORGE_HOME` + localhost ports
- Starts/stops Core (and optional SeaweedFS if you provide a `weed` binary under `${FACEFORGE_HOME}/tools`)
- Runs in the tray (closing the window hides it)

Notes:

- Desktop launches Core using the repo-local `.venv` if present. If you haven't bootstrapped `.venv` yet, run `./scripts/dev-core.ps1` once.
- "Open UI" opens `/ui/login` (paste token once; it becomes a cookie).

Run Core (dev)

From the repo root (Windows PowerShell):

Prereq: Python 3.12.x installed (used only to bootstrap the repo-local `.venv`).

- `./scripts/dev-core.ps1`

Then open:

- `http://127.0.0.1:8787/healthz`
- `http://127.0.0.1:8787/docs`

Core also serves a lightweight **Web UI** (no runtime Node dependency):

- `http://127.0.0.1:8787/` (redirects to the UI)
- `http://127.0.0.1:8787/ui/login`

Core now exposes initial Entities + Assets v1 endpoints (token required):

- `GET /v1/entities`
- `POST /v1/entities`
- `GET/PATCH/DELETE /v1/entities/{entity_id}`
- `POST /v1/assets/upload` (multipart)
- `GET /v1/assets/{asset_id}`
- `GET /v1/assets/{asset_id}/download` (streaming + HTTP Range)
- `POST /v1/assets/bulk-import` (starts a durable job)
- `POST /v1/entities/{entity_id}/assets/{asset_id}` (link)
- `DELETE /v1/entities/{entity_id}/assets/{asset_id}` (unlink)

Core also exposes initial Jobs endpoints (Sprint 9):

- `POST /v1/jobs`
- `GET /v1/jobs`
- `GET /v1/jobs/{job_id}`
- `GET /v1/jobs/{job_id}/log` (append-only, pollable)
- `POST /v1/jobs/{job_id}/cancel` (cooperative cancellation)

Core also exposes initial Descriptor + Field Definition endpoints (Sprint 7):

- `GET /v1/admin/field-defs` (admin)

- `POST /v1/admin/field-defs` (admin)
- `GET /v1/admin/field-defs/{field_def_id}` (admin)
- `PATCH /v1/admin/field-defs/{field_def_id}` (admin)
- `DELETE /v1/admin/field-defs/{field_def_id}` (admin)
- `GET /v1/entities/{entity_id}/descriptors`
- `POST /v1/entities/{entity_id}/descriptors`
- `PATCH /v1/descriptors/{descriptor_id}`
- `DELETE /v1/descriptors/{descriptor_id}`

Core also exposes initial Relationships endpoints (Sprint 8):

- `POST /v1/relationships`
- `GET /v1/relationships?entity_id=...`
- `DELETE /v1/relationships/{relationship_id}`
- `GET /v1/relation-types?query=...`

Core also exposes initial Plugins endpoints:

- `GET /v1/plugins`
- `POST /v1/plugins/{plugin_id}/enable`
- `POST /v1/plugins/{plugin_id}/disable`
- `GET /v1/plugins/{plugin_id}/config`
- `PUT /v1/plugins/{plugin_id}/config`

Plugins

Core discovers plugin manifests from:

- `${FACEFORGE_HOME}/plugins/*/plugin.json`

Minimal `plugin.json` example:

```json
{
  "id": "demo.plugin",
  "name": "Demo Plugin",
  "version": "0.1.0",
  "capabilities": ["ui"],
  "config_schema": {
    "type": "object",
    "properties": {
      "example_flag": { "type": "boolean" }
    },
    "additionalProperties": true
  }
}
```

Notes:

- Plugin enable/disable and config are persisted in Core's SQLite DB.
- If a `config_schema` is provided, `PUT /v1/plugins/{plugin_id}/config` validates the config using JSON Schema.
- The namespace `/v1/plugins/{plugin_id}/...` is reserved for future plugin-provided HTTP surfaces.

For optional SeaweedFS/S3 storage configuration (Sprint 6), see:

- core/README.md

Auth

Core binds to localhost by default and requires a **per-install token** for non-health endpoints.

- Health endpoint (no token): `GET /healthz`
- Example protected endpoint: `GET /v1/ping`
- System identity endpoint (protected): `GET /v1/system/info`

Build a Windows executable (Core)

To produce a standalone `faceforge-core.exe` (PyInstaller one-file build):

1. Verify Python is available (3.12.x):
   - `python --version`
2. Build the executable from the repo root:
   - `./scripts/build-core.ps1`
   - This script creates/updates the repo-local `.venv`, installs build dependencies, and runs PyInstaller.
   - If it fails to delete `core/dist` or `core/build`, close any terminals/Explorer windows using those folders and re-run.
     - Optional: `./scripts/build-core.ps1 -AllowTimestampFallback`

Outputs:

- `core/dist/faceforge-core.exe`

Run the built executable (example):

- `./core/dist/faceforge-core.exe --help`

Notes:

- Core still uses `FACEFORGE_HOME` as its data root; if it’s not set, Core will use the default behavior described in the spec.
- On first start, Core generates the install token (if missing) and writes it under `${FACEFORGE_HOME}/config/core.json`.

If you want to run Core in dev instead of building an exe:

- `./scripts/dev-core.ps1`

GitHub Releases

This repo includes a GitHub Actions workflow that builds and attaches release assets automatically.

- Workflow: `.github/workflows/release-core.yml`
- Trigger: publish a GitHub Release for a tag like `v0.1.0` (or run the workflow manually with a tag input)

How to cut a release (GitHub UI):

1. Go to **Releases** → **Draft a new release**.
2. Pick an existing tag or create a new one (recommended tag format: `v0.1.0`).
3. Fill in title/notes and click **Publish release**.
4. Wait for the **release-core** workflow to finish; it will upload assets onto the Release.

After the workflow completes, the Release will contain:

- `faceforge-core.exe` (Windows)
- `faceforge-core.sha256` (Windows)
- `faceforge-core-*.whl` (Python wheel)
- `faceforge-core-*.tar.gz` (Python sdist)

The token is generated on first start (if missing) and stored in:

- `${FACEFORGE_HOME}/config/core.json` → `auth.install_token`

Send it using either header:

- `Authorization: Bearer <token>`
- `X-FaceForge-Token: <token>`

For the Web UI in a browser, you can also log in at `/ui/login`, which stores the token in an HttpOnly cookie.

Example (PowerShell):

- `Invoke-RestMethod -Headers @{ Authorization = "Bearer <token>" } http://127.0.0.1:8787/v1/ping`

Checks (format + lint + tests)

- `./scripts/check-core.ps1`

These scripts create/use the repo-local `.venv` and always run commands through it.

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 doc via the repo UI: copilot-instructions.md

Building for Release

To create a distributable installer:

1. **Build Core Executable**:

   ```
   ./scripts/build-core.ps1
   ```

   This uses PyInstaller to freeze the Python service into `core/dist/faceforge-core.exe`.

2. **Build Desktop Bundle**:

   ```
   cd desktop
   npm install
   npm run tauri build
   ```

   This packages the Desktop shell, bundles the Core executable (as a sidecar), and includes necessary tools (ExifTool, etc.).

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.