FaceForge Core

This folder contains the **Core API service** (planned: FastAPI + Uvicorn).

Repo structure (Core)

If you're trying to find things quickly:

```
core/
    src/faceforge_core/
        __main__.py           # `python -m faceforge_core` entrypoint
        app.py                # FastAPI app wiring
        api/v1/               # versioned HTTP routes
        auth.py               # install-token auth helpers
        config.py             # config loading + path resolution
        home.py               # FACEFORGE_HOME + layout contract
        db/                   # SQLite schema + migrations + queries
        internal/             # internal CLIs (dev/admin helpers)
    tests/                  # pytest tests
    pyproject.toml          # packaging/dependencies
```

Sprint 1: home/config/runtime contract

Core is **local-first** and writes all persistent/runtime files under `FACEFORGE_HOME`.

FACEFORGE_HOME

- If `FACEFORGE_HOME` is set, Core uses that directory.
- If it is not set, Core defaults to a per-user OS data directory (never relative to the working directory):
    - Windows: `%LOCALAPPDATA%\\FaceForge`
    - macOS: `~/Library/Application Support/FaceForge`
    - Linux: `$XDG_DATA_HOME/faceforge` (or `~/.local/share/faceforge`)

Required subfolders

On startup, Core ensures these directories exist:

- `${FACEFORGE_HOME}/db`
- `${FACEFORGE_HOME}/s3`
- `${FACEFORGE_HOME}/assets`
- `${FACEFORGE_HOME}/logs`

- `${FACEFORGE_HOME}/tmp`
- `${FACEFORGE_HOME}/config`
- `${FACEFORGE_HOME}/tools`
- `${FACEFORGE_HOME}/plugins`

Core config file

Core loads JSON config from:

- `${FACEFORGE_HOME}/config/core.json`

If the file does not exist, defaults are used.

Current config shape (v1, subject to change):

```
{
    "version": "1",
    "auth": {
        "install_token": "..."
    },
    "network": {
        "bind_host": "127.0.0.1",
        "core_port": 8787,
        "seaweed_s3_port": null
    },
    "paths": {
        "db_dir": null,
        "s3_dir": null,
        "logs_dir": null,
        "plugins_dir": null
    },
    "tools": {
        "exiftool_enabled": true,
        "exiftool_path": null
    },
    "storage": {
        "routing": {
            "default_provider": "fs",
            "kind_map": {},
            "s3_min_size_bytes": null
        },
        "s3": {
            "enabled": false,
            "endpoint_url": null,
            "access_key": null,
            "secret_key": null,
            "bucket": "faceforge",
            "region": "us-east-1",
            "use_ssl": false
        }
    },
    "seaweed": {
        "enabled": false,
        "weed_path": null,
        "data_dir": null,
        "ip": "127.0.0.1",
        "master_port": 9333,
        "volume_port": 8080,
        "filer_port": 8888,
        "s3_port": null
    }
}
```

Notes:

- `paths.*` may be absolute paths or paths relative to `FACEFORGE_HOME`.
- `tmp/` and `config/` are intentionally **not configurable**.

Runtime ports file

Desktop (or other launcher) may write the selected ports to:

- `${FACEFORGE_HOME}/config/ports.json`

Format:

```
{
    "core": 43210,
    "seaweed_s3": 43211
}
```

Core also supports a legacy location for compatibility with the design spec:

- `${FACEFORGE_HOME}/runtime/ports.json`

Desktop integration (Sprint 12)

FaceForge Desktop (Tauri) is responsible for the first-run wizard and process orchestration.

Desktop writes (or updates):

- `${FACEFORGE_HOME}/config/ports.json` (selected ports)
- `${FACEFORGE_HOME}/config/core.json` (network + auth token; optional S3 config)

Desktop then starts Core with:

- `FACEFORGE_HOME` set to the chosen directory
- `FACEFORGE_BIND=127.0.0.1` (localhost-only by default)

For browser usage, the Core UI requires logging in once at `/ui/login` (pastes the install token and stores it in an HttpOnly cookie).

Dev run

From the repo root (Windows PowerShell):

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

- `./scripts/check-core.ps1` (format + lint + tests)
- `./scripts/build-core.ps1` (builds a local `core/dist/faceforge-core.exe`)

Build outputs note:

- `./scripts/build-core.ps1` will automatically prune old timestamped `core/build-*` and `core/dist-*` folders left behind by prior locked builds.
- If you want to keep build history for debugging, run: `./scripts/build-core.ps1 -KeepBuildHistory`
- If `core/build` or `core/dist` cannot be deleted (e.g. you have a terminal opened inside those folders), the build now fails with a clear message instead of creating timestamped folders. To opt into the old fallback behavior, run: `./scripts/build-core.ps1 -AllowTimestampFallback`

This repo is set up to avoid relying on a global Python for running commands. The scripts create/use the repo-local `.venv` and always run via `.venv\\Scripts\\python.exe`.

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

The service should come up on `http://127.0.0.1:8787` and expose:

- `GET /healthz`
- `GET /docs` (public)
- `GET /` (redirects to Web UI)

Sprint 11: Core Web UI MVP

Core serves a small, server-rendered Web UI (no runtime Node) intended for basic, non-technical workflows.

Routes:

- `GET /ui/login` (public): paste the install token and sign in
- `POST /ui/login`: sets an HttpOnly cookie
- `POST /ui/logout`: clears the cookie
- `GET /ui/entities`: browse/create entities (table/gallery)
- `GET /ui/entities/{entity_id}`: entity detail (overview/descriptors/attachments/relationships)
- `GET /ui/jobs`: job list + start bulk-import
- `GET /ui/jobs/{job_id}`: job details + logs
- `GET /ui/plugins`: list discovered plugins, enable/disable, edit config

API endpoints (v1):

- `GET /v1/ping` (requires token)

- `GET /v1/system/info` (requires token)
- `GET /v1/entities` (requires token)
- `POST /v1/entities` (requires token)
- `GET/PATCH/DELETE /v1/entities/{entity_id}` (requires token)
- `POST /v1/assets/upload` (requires token; multipart)
- `GET /v1/assets/{asset_id}` (requires token)
- `GET /v1/assets/{asset_id}/download` (requires token; streaming + Range)
- `POST /v1/assets/bulk-import` (requires token; starts a job)
- `POST /v1/entities/{entity_id}/assets/{asset_id}` (requires token; link)
- `DELETE /v1/entities/{entity_id}/assets/{asset_id}` (requires token; unlink)
- `POST /v1/jobs` (requires token)
- `GET /v1/jobs` (requires token)
- `GET /v1/jobs/{job_id}` (requires token)
- `GET /v1/jobs/{job_id}/log` (requires token; pollable)
- `POST /v1/jobs/{job_id}/cancel` (requires token)
- `GET /v1/admin/field-defs` (requires token)
- `POST /v1/admin/field-defs` (requires token)
- `GET /v1/admin/field-defs/{field_def_id}` (requires token)
- `PATCH /v1/admin/field-defs/{field_def_id}` (requires token)
- `DELETE /v1/admin/field-defs/{field_def_id}` (requires token)
- `GET /v1/entities/{entity_id}/descriptors` (requires token)
- `POST /v1/entities/{entity_id}/descriptors` (requires token)
- `PATCH /v1/descriptors/{descriptor_id}` (requires token)
- `DELETE /v1/descriptors/{descriptor_id}` (requires token)
- `POST /v1/relationships` (requires token)
- `GET /v1/relationships?entity_id=...` (requires token)
- `DELETE /v1/relationships/{relationship_id}` (requires token)
- `GET /v1/relation-types?query=...` (requires token)

- `GET /v1/plugins` (requires token)
- `POST /v1/plugins/{plugin_id}/enable` (requires token)
- `POST /v1/plugins/{plugin_id}/disable` (requires token)
- `GET /v1/plugins/{plugin_id}/config` (requires token)
- `PUT /v1/plugins/{plugin_id}/config` (requires token)

Auth (Sprint 3)

Core requires a per-install token for non-health endpoints.

- The token is stored in `${FACEFORGE_HOME}/config/core.json` under `auth.install_token`.
- Requests may provide the token via:
  - `Authorization: Bearer <token>`
  - `X-FaceForge-Token: <token>`

Browser UI support:

- The Web UI can store the token in an HttpOnly cookie named `ff_token` (set by `POST /ui/login`).
- This makes browser downloads (e.g. `GET /v1/assets/{asset_id}/download`) work without manually setting headers.

Sprint 10: Plugins (discovery + registry)

Core discovers plugin manifests under `${FACEFORGE_HOME}/plugins/*/plugin.json` and stores metadata/config in the `plugin_registry` table.

The v1 endpoints under `/v1/plugins` expose discovery results, enable/disable state, and a JSON Schema-validated config document.

Manifest format (`plugin.json`)

Minimal example:

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

Supported fields today (best-effort discovery):

- `id` (required): stable plugin identifier.
- `name` (required)
- `version` (optional)
- `capabilities` (optional list)
- `config_schema` (optional object): JSON Schema used to validate `PUT /v1/plugins/{plugin_id}/config`.

Notes:

- Discovery is best-effort: invalid manifests are ignored.
- `/v1/plugins/{plugin_id}/...` is reserved for future plugin routes (plugin compute is not implemented in Core).

Sprint 2: SQLite schema + migrations (internal)

Core stores metadata in a SQLite DB under:

- `${FACEFORGE_HOME}/db/core.sqlite3`

On Core startup, schema migrations are applied automatically (idempotent).

Apply migrations and create sample records without using the API:

1) Run `./scripts/dev-core.ps1` once (it installs Core editable into `.venv`). 2) Then invoke internal modules using the venv Python:

- `.\.venv\Scripts\python.exe -m faceforge_core.internal.bootstrap_db --home <PATH> --migrate`
- `.\.venv\Scripts\python.exe -m faceforge_core.internal.bootstrap_db --home <PATH> --create-entity "Ada Lovelace"`
- `.\.venv\Scripts\python.exe -m faceforge_core.internal.bootstrap_db --home <PATH> --create-asset <FILEPATH>`

Sprint 4: Entities CRUD (v1)

Endpoints:

- `GET /v1/entities`
- `POST /v1/entities`
- `GET /v1/entities/{entity_id}`
- `PATCH /v1/entities/{entity_id}`
- `DELETE /v1/entities/{entity_id}` (soft delete)

List query params (minimal primitives):

- `limit` (default 50, max 200)
- `offset` (default 0)
- `sort_by`: `created_at` | `updated_at` | `display_name`
- `sort_order`: `asc` | `desc`
- `q`: substring match (basic)
- `tag`: filter by exact tag string

Sprint 5: Assets v1 (filesystem provider)

Upload

- `POST /v1/assets/upload`
  - Multipart field `file` (required)
  - Multipart field `meta` (optional): companion JSON sidecar (commonly `_meta.json`)

Example (PowerShell):

- `Invoke-WebRequest -Headers @{ Authorization = "Bearer <token>" } -Form @{ file = Get-Item .\myfile.bin; meta = Get-Item .\_meta.json } http://127.0.0.1:8787/v1/assets/upload`

Metadata

- `GET /v1/assets/{asset_id}`

Download (streaming + resume)

- `GET /v1/assets/{asset_id}/download`
- Supports `Range: bytes=...` (single range)

Example (curl):

- `curl -H "Authorization: Bearer <token>" -H "Range: bytes=0-1048575" -o first-1mb.bin http://127.0.0.1:8787/v1/assets/<asset_id>/download`

Sprint 9: Jobs + structured logs + bulk import

Core exposes a minimal durable job model backed by SQLite:

- Jobs are created with status `queued` and executed in-process.
- Logs are append-only and stored in `job_logs` with timestamps and levels.

- Cancellation is cooperative: `POST /v1/jobs/{job_id}/cancel` requests cancellation and the worker stops at a safe point.

Create a bulk import job

- `POST /v1/assets/bulk-import`
  - Body: `{ "path": "...", "recursive": true, "kind": "file" }`
  - Response: `{ job_id, job_type, status, created_at }`

Then poll:

- `GET /v1/jobs/{job_id}` (status + progress)
- `GET /v1/jobs/{job_id}/log?after_id=0` (logs; use `next_after_id` for incremental polling)

Link/unlink to entities

- `POST /v1/entities/{entity_id}/assets/{asset_id}` (optional JSON body: `{ "role": "..." }`)
- `DELETE /v1/entities/{entity_id}/assets/{asset_id}`

ExifTool metadata extraction (best-effort)

On upload, Core will attempt to run ExifTool in the background (if available) and store extracted JSON under the asset `meta.metadata[]` list.

Configuration:

- `${FACEFORGE_HOME}/config/core.json` → `tools.exiftool_enabled` (default `true`)
- `${FACEFORGE_HOME}/config/core.json` → `tools.exiftool_path` (optional override)

Sprint 7: Descriptors + Field Definitions (admin)

FaceForge Core supports a small “flexible schema without migrations” layer:

- Admin creates **field definitions** (scope + key + type + validation rules)
- API accepts **descriptors** immediately once a field definition exists
- Invalid descriptor values are rejected with a clear `validation_error` response

Field definitions

- `GET /v1/admin/field-defs` (optional query: `scope=…`)
- `POST /v1/admin/field-defs`

- `GET /v1/admin/field-defs/{field_def_id}`
- `PATCH /v1/admin/field-defs/{field_def_id}`
- `DELETE /v1/admin/field-defs/{field_def_id}` (soft delete)

Field definition fields:

- `scope` (string; default `descriptor`)
- `field_key` (string)
- `field_type` (string; supports `string`, `int`, `float`, `bool`, `enum`, `json`)
- `required` (bool)
- `regex` (string; `string` only)
- `options` (object; for `enum`, use `{ "options": ["a", "b"] }`)

Descriptors

- `GET /v1/entities/{entity_id}/descriptors`
- `POST /v1/entities/{entity_id}/descriptors`
- `PATCH /v1/descriptors/{descriptor_id}`
- `DELETE /v1/descriptors/{descriptor_id}` (soft delete)

Descriptors are validated against the matching field definition (`scope` + `field_key`).

Bundling/embedding requirement:

- Core does **not** search your system `PATH` for `exiftool`.
- If `tools.exiftool_path` is not set, Core only checks bundled locations under `${FACEFORGE_HOME}/tools`, using these candidate paths:
  - Windows: `${FACEFORGE_HOME}/tools/exiftool.exe` or `${FACEFORGE_HOME}/tools/exiftool/exiftool.exe`
  - macOS/Linux: `${FACEFORGE_HOME}/tools/exiftool` or `${FACEFORGE_HOME}/tools/exiftool/exiftool`

Sprint 6: SeaweedFS provider + “default local S3” wiring

Core can optionally store asset bytes in an **S3-compatible backend** (intended: SeaweedFS S3 endpoint), while still supporting filesystem-only mode.

Storage routing (upload-time)

Uploads are routed based on:

- `storage.routing.kind_map` (highest priority)
- `storage.routing.s3_min_size_bytes` (optional)

- `storage.routing.default_provider`

If routing selects `s3` but the endpoint is not reachable, Core **falls back to filesystem** automatically.

S3 configuration

Set these in `${FACEFORGE_HOME}/config/core.json`:

- `storage.s3.enabled`: enable S3 provider
- `storage.s3.endpoint_url`: optional; if omitted, Core derives from `network.bind_host` + `network.seaweed_s3_port`
- `storage.s3.access_key` / `storage.s3.secret_key`: credentials for the endpoint
- `storage.s3.bucket`: default bucket name (Core will create it best-effort)

SeaweedFS “managed binary” contract (optional)

Desktop will orchestrate SeaweedFS later, but Core can optionally start it for dev/testing if you set `seaweed.enabled=true` and provide the `weed` binary under `${FACEFORGE_HOME}/tools`.

Binary resolution order:

- `seaweed.weed_path` (absolute or relative to `${FACEFORGE_HOME}/tools`)
- Otherwise, Core checks:
  - Windows: `${FACEFORGE_HOME}/tools/weed.exe`, `${FACEFORGE_HOME}/tools/seaweedfs/weed.exe`, `${FACEFORGE_HOME}/tools/seaweed/weed.exe`
  - Non-Windows: analogous paths without `.exe`

Core-managed runner:

- `weed server -s3 …` with ports from `seaweed.*` (and/or `network.seaweed_s3_port`)
- Data dir defaults to `${FACEFORGE_HOME}/s3/seaweedfs` (override via `seaweed.data_dir`)

Dev helper CLI:

- `.\.venv\Scripts\python.exe -m faceforge_core.internal.seaweedfs_cli --home <FACEFORGE_HOME> --health`
- `.\.venv\Scripts\python.exe -m faceforge_core.internal.seaweedfs_cli --home <FACEFORGE_HOME> --run`
- If `tools.exiftool_path` is a relative path, it is resolved relative to `${FACEFORGE_HOME}/tools`.

Core skips ExifTool processing for filenames matching the exclusions defined in core/src/faceforge_core/ingest/exiftool.py.