

FaceForge Core - MVP Sprint Sequence

Below is a sprint sequence that gets us to a shippable **FaceForge Core MVP** without any single sprint turning into a swamp. Each sprint is scoped so you can close it cleanly, tag a milestone, and start the next session with a fresh Copilot thread.

SUMMARY CONCEPTS

MVP Definition of "Done"

A non-developer can install/run **FaceForge Desktop**, pick `FACEFORGE_HOME`, and then:

- open the local web UI (served by Core)
- create/edit entities
- upload assets, link them to entities, download them (streaming + resume/range)
- run at least one real job (bulk import) and view job logs
- enable/disable a “plugin” (manifest discovery + config surface; plugin compute not required)
- everything binds to localhost by default with a per-install token

(Aligned with the v0.2.9 spec.)

Optional “buffer” sprints (only if you feel pain)

- **Perf & indexing pass:** SQLite indexes for common list queries, generated columns for frequently filtered JSON fields
- **Backup/export v0:** simple metadata export + asset listing (even if FFBackup becomes a plugin later)
- **API polish:** bulk-upsert endpoint and conflict rules, if you need it early

Next Steps

If you want to run this like a true sprint board, the cleanest next move is to turn each sprint into an epic, then break deliverables into 5–15 bite-sized issues each (small enough that Copilot doesn’t melt down mid-session).

THE SPRINT SEQUENCE

Sprint 0 — Repo + “boring core” scaffolding

- **Objective:** Make the project easy to work on before you write real features.
- **Deliverables**
 - Monorepo structure (or clean multi-repo) with:
 - `core/` (FastAPI service)
 - `desktop/` (Tauri)

- `docs/` (dev + user docs)
 - Dev bootstrap scripts (one command to run Core in dev, one for UI dev if separate)
 - Formatting/linting/test harness wired (pre-commit or equivalent)
 - CI pipeline: lint + unit tests + build artifacts (at least Core)
 - **Acceptance**
 - Fresh clone → one documented command starts Core + a placeholder `/healthz`
 - CI runs on PRs without manual babysitting
-

Sprint 1 — Config + filesystem layout contract

- **Objective:** Lock in how the app finds its home and writes files.
 - **Deliverables**
 - `FACEFORGE_HOME` resolution rules + defaults
 - Directory creation rules (`db/`, `s3/`, `logs/`, `run/`, `config/`, `plugins/`)
 - Core config file format (JSON suggested in spec) + loader + validation
 - Runtime “ports.json” convention (even if Desktop owns it later)
 - **Acceptance**
 - Core starts with only `FACEFORGE_HOME` set and creates required subfolders
 - Config changes require no code changes (just file edits)
-

Sprint 2 — SQLite schema v1 + ID strategy

- **Objective:** Build the relational spine cleanly before endpoints pile up.
- **Deliverables**
 - SQLite schema + migrations strategy (simple migration runner is fine)
 - Tables for:
 - `entities`, `assets`, `entity_assets` (many-to-many)
 - `relationships`
 - `jobs`, `job_logs`
 - `field_definitions`
 - `plugin_registry` (enabled/config/version snapshot)
 - SHA-256 ID utilities:
 - entity IDs generated at creation time
 - asset IDs + `content_hash` computed from bytes
 - Soft delete strategy decided (and implemented where relevant)
- **Acceptance**
 - “Create entity” + “create asset record” can be done via a tiny internal script without API

- Migration runner works from blank → latest
-

Sprint 3 — Core API skeleton + auth defaults

- **Objective:** A real API surface that won't need rework later.
 - **Deliverables**
 - FastAPI app structure with versioned routing (`/v1/...`)
 - Token auth middleware (per-install token stored in config; localhost by default)
 - Standard response envelope + error model (consistent errors matter)
 - `/docs` and `/redoc` enabled and accurate
 - **Acceptance**
 - A request without token fails (except health endpoints you explicitly exempt)
 - OpenAPI renders and matches real routes
-

Sprint 4 — Entities CRUD + list/search primitives

- **Objective:** The simplest "real feature" end-to-end.
 - **Deliverables**
 - Endpoints:
 - `GET /v1/entities`
 - `POST /v1/entities`
 - `GET/PATCH/DELETE /v1/entities/{entity_id}`
 - Minimal filtering/paging/sorting (don't overbuild search yet)
 - Entity model supports: `display_name`, `aliases`, `tags`, `fields` JSON, timestamps
 - **Acceptance**
 - You can create 100 entities and page through them
 - Patch updates only touched fields; timestamps behave sensibly
-

Sprint 5 — Assets v1: upload, metadata, link/unlink, download (filesystem provider)

- **Objective:** Make file ingest + retrieval rock-solid before adding S3.
- **Deliverables**
 - Storage abstraction interface + **local filesystem provider** implementation
 - `POST /v1/assets/upload` (multipart; optional companion `_meta.json`)
 - `GET /v1/assets/{asset_id}` metadata
 - Link/unlink endpoints to entities
 - `GET /v1/assets/{asset_id}/download`:
 - streaming response
 - HTTP range support (resume-friendly)

- **IMPORTANT:** Ingest metadata extraction via `exiftool`

- Embed `exiftool` binary (avoid Perl requirements) - install the correct edition per OS (this should be incorporated into the install process later)
- Run on asset upload to extract metadata
 - Store in asset record under `metadata` JSON field
 - Spawn parallel process to run `exiftool` if needed (be sure to log events properly)
 - EXCEPTIONS (Files matching these patterns should skip exiftool processing):
 - `_meta|directorymeta|.json$`
 - `\.(cover|thumb|thumb(s|db|index|nail))$`
 - `^(thumb|thumb(s|db|index|nail))\.db$`
- Send parameters to exiftool via parameter file: `exiftool -@ "$ArgsFile" 2> $null` (in PowerShell, for example)
- Use these exact parameters in the `$ArgsFile`:

```
-quiet -extractEmbedded3 -scanForXMP -unknown2 -json -G3:1 -struct
-b -ignoreMinorErrors -charset filename=utf8 -api requestall=3 -
api largefilesupport=1 --
```

- Parse the output and **REMOVE** the following keys from the JSON before storing:

- `ExifTool:ExifToolVersion`
- `ExifTool:FileSequence`
- `ExifTool:NewGUID`
- `System:BaseName`
- `System:Directory`
- `System:FileBlockCount`
- `System:FileBlockSize`
- `System:FileDeviceID`
- `System:FileDeviceNumber`
- `System:FileGroupID`
- `System:FileHardLinks`
- `System:FileInodeNumber`
- `System:FileName`
- `System:FilePath`
- `System:FilePermissions`
- `System:FileUserID`

- The final resulting JSON data (`$ExifToolOutput`) must be validated as proper JSON and being not-empty and nested into the following structure before storing:

```
{
  "Source": "ExifTool",
  "Type": "JsonMetadata",
  "Name": $null,
  "NameHashes": $null,
  "Data": {ExifToolOutput}
}
```

- **Acceptance**

- Upload a 2–5GB file and download it reliably
 - Range requests work (spot-check with curl / a download manager)
 - Uploaded asset gets metadata extracted and stored correctly (with proper error handling/logging/validation)
-

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

- **Objective:** Bring the spec’s default object storage online without dragging Desktop into it yet.

- **Deliverables**

- SeaweedFS “managed binary” contract defined (paths, args, health checks)
- S3-compatible provider implementation via SeaweedFS endpoint
- Storage routing rules (by asset **kind** or size threshold) working
- Core can operate even if SeaweedFS is disabled (fallback to filesystem provider)

- **Acceptance**

- With SeaweedFS running: uploads land in the S3 backend
 - With SeaweedFS off: uploads still work (filesystem)
-

Sprint 7 — Descriptors + Field Definitions (admin)

- **Objective:** Get the “flexible schema without migrations” concept working.

- **Deliverables**

- Field definition endpoints:
 - **GET/POST/PATCH/DELETE /v1/admin/field-defs...**
- Descriptor endpoints:
 - **GET/POST /v1/entities/{entity_id}/descriptors**
 - **PATCH/DELETE /v1/descriptors/{descriptor_id}**
- Validation driven by field definitions (at least type + required + regex/options)

- **Acceptance**

- Add a new field definition → UI/API accepts it immediately
 - Invalid descriptor values are rejected with clear errors
-

Sprint 8 — Relationships + relation type suggestion

- **Objective:** Relationship metadata, without graph visualization.

- **Deliverables**

- **POST /v1/relationships** and **GET /v1/relationships?entity_id=...**
- **DELETE /v1/relationships/{relationship_id}**

- `GET /v1/relationship-types?query=...` backed by:
 - existing relationship types in DB
 - a small built-in seed list (so first-time UX isn't empty)
 - **Acceptance**
 - Create relationship types organically; suggestion endpoint surfaces them
 - Relationship records round-trip correctly and are queryable
-

Sprint 9 — Jobs + structured logs + one “real” job

- **Objective:** A durable job model that plugins can piggyback on later.
 - **Deliverables**
 - `POST /v1/jobs`, `GET /v1/jobs/{job_id}`, `GET /v1/jobs/{job_id}/log`, `POST /v1/jobs/{job_id}/cancel`
 - Append-only structured logging with timestamps + levels
 - Progress reporting fields (percent + step name)
 - Implement **one Core job** end-to-end:
 - `assets/bulk-import` as a job (directory scan + sidecar `_meta.json`)
 - **Acceptance**
 - Start bulk import → watch progress/logs live → job completes
 - Cancellation behaves predictably (stops work, marks state)
-

Sprint 10 — Plugin foundations v1 (discovery + registry + config surface)

- **Objective:** “Plugins exist” without building plugin compute yet.
 - **Deliverables**
 - Plugin manifest format (`plugin.json`) and loader
 - Discovery from `#{FACEFORGE_HOME}/plugins`
 - Registry persisted in DB:
 - enabled/disabled
 - stored config blob (validated against JSON schema if provided)
 - API:
 - `GET /v1/plugins`
 - enable/disable
 - `GET/PUT /v1/plugins/{id}/config`
 - Namespacing rules established (`/v1/plugins/<id>/...` reserved)
 - **Acceptance**
 - Drop a manifest into plugins dir → it appears in API/UI
 - Enable/disable state persists across restarts
-

Sprint 11 — Core Web UI MVP (Core-only features)

- **Objective:** A usable UI that covers everything Core can do (no “backend-only” gaps).
 - **Deliverables**
 - Pages:
 - Entities list (table + gallery toggle)
 - Entity detail with tabs: Overview / Descriptors / Attachments / Relationships
 - Jobs page: list + detail log view
 - Plugins page: list + enable/disable + config form rendering (basic)
 - Attachments UX:
 - upload
 - link/unlink
 - download
 - Lightweight styling, fast load, no runtime Node dependency
 - **Acceptance**
 - A non-technical user can do the whole “create entity → upload asset → link → download” loop without touching API tools
 - Jobs are viewable and understandable from UI
-

Sprint 12 — Desktop shell MVP (Tauri orchestrator + tray UX)

- **Objective:** Make it run like a real app, not a repo full of scripts.
 - **Deliverables**
 - First-run wizard:
 - choose `FACEFORGE_HOME`
 - choose/auto-pick Core port (and SeaweedFS port if needed)
 - Process orchestration:
 - start/stop Core service
 - start/stop SeaweedFS (if enabled)
 - basic health monitoring + restart strategy
 - Tray behavior:
 - close to tray (not exit)
 - menu: Open UI, Status, Logs, Stop, Restart, Exit
 - on exit: prompt to stop services or leave running
 - **Acceptance**
 - Double-click app → wizard → “Open UI” works
 - Closing window leaves tray running; Exit performs expected shutdown behavior
-

Sprint 13 — Packaging + release hardening

- **Objective:** Turn “works on my machine” into “ship it.”

- **Deliverables**

- Packaging plan implemented:
 - Core service shipped as frozen binary or embedded Python runtime (pick one and commit)
 - SeaweedFS binaries bundled or fetched in a controlled way
- Installer / portable bundle per OS (Windows first if that's your priority)
- Logging + crash reports in `#{FACEFORGE_HOME}/logs`
- Security defaults verified:
 - localhost bind by default
 - token required
 - config/secrets not accidentally committed
- "First-run screenshots + quickstart" user docs

- **Acceptance**

- Clean machine install → user completes wizard → app works without dev tooling
 - Basic smoke test checklist passes reliably
-