

TECH SPEC EN — Audion Hub Manager

Contents

- [1. Purpose](#)
- [2. Architecture Principle](#)
- [2.1. Project Registry](#)
- [3. MIRROR Engine](#)
- [4. Projection Profiles](#)
- [5. UI Layout](#)
- [6. Core Panes](#)
- [7. Terminal Dock](#)
- [8. Git Engine](#)
- [9. Auth Policy](#)
- [10. Storage Policy](#)
- [11. Testing](#)
- [12. Documentation Artifacts](#)

1. Purpose

Audion Hub Manager is a local commander interface for:

```
Full Project -> Hub Projection -> Git -> GitHub/GitLab/Codeberg/local bundle
      ^
      VS Code / Docs layer / NiceGUI UI
```

It does not replace VS Code and does not bind the user to Obsidian, LogSeq or any other Markdown tool. It makes MIRROR, Git state, diff, commit, safety scan and checkpoint flow visible in one workbench.

2. Architecture Principle

Project first.
Hub follows.

Full Project is the source of truth. Hub Data is a filtered technical projection. Docs is an optional human-readable view.

Hub does not invent documentation. It mirrors what already exists in the project: `README`, `TECH_SPEC`, `AGENTS`, `CHANGELOG`, `docs/`, `prompts/`, `reports/` and allowed code/config files.

2.1. Project Registry

`config/projects.json` describes a set of separate projects, not one large Source container. The `Project` dropdown switches the whole active entry:

```
source_path      -> Project layer
projection_path  -> Mirror / Hub Data layer
docs_path       -> Docs layer
profile         -> MIRROR rules and commit allowlist
default_branch  -> branch used by Git commands
```

The `Project` layer above the tree shows the active entry's `source_path`. This is intentionally distinct from a broad Source parent folder that may contain many projects.

`Scan projects` scans a parent folder with many projects, detects real roots through project markers and developed programming-language file structure, skips launcher-only `.cmd` folders, and appends missing entries to `projects.json`. Hub/Data and Docs branches inside the scan area must be skipped.

`Clean projects.json` removes only registry records with missing `source_path` values and duplicates. It lives in Support, writes a Storage/terminal report, and never deletes Source, Hub Data, Docs or other project folders.

3. MIRROR Engine

Base module:

```
system_core/core/projection_engine.py
```

Core algorithm:

```
scan source by profile
scan target by profile
compare relative paths
copy/update/touch first
delete projection-only stale files only after clean copy phase
sync directory shape
ensure .gitkeep in empty incoming dirs
```

Rules:

- Source is not modified by MIRROR.
- Projection may delete stale files.
- `.git/**` is protected.
- Real apply requires explicit intent.
- Filtered profile must have a non-empty include/allowlist.

4. Projection Profiles

Profiles live in:

```
config/projection_profiles.json
```

The Dev-Git profile should include:

- source files;
- Markdown/text documentation;
- lock files and reproducibility manifests;
- CI/build descriptors;
- formatter/linter configs;
- useful shared `.vscode/tasks.json`, `launch.json`, `extensions.json`.

The profile should exclude:

- runtime payload;

- build/dist/out/target;
- caches;
- logs;
- binaries/media/archives;
- `.env`, private keys, tokens;
- machine-local overrides such as `*.local.json`.

5. UI Layout

Primary layout:

Control | Structure | Work Area
Bottom: Terminal Dock / Command Input / Command Cache

Work Area right tabs, arranged as two semantic toggle rows:

Quick | Branch | Editor | Diff | Storage
Remote | Basket | Reader | History | Details

Each right tab header starts with a Material icon. Command buttons and right tab headers expose tooltips with a 1200 ms show delay and 80 ms hide/fade.

Panes should not be packed into one generic inspector. Each pane has its own work role.

The left panel is split into:

- **Open** — open the current Project/Source, Mirror, Docs Folder, VS Code, Terminal and Git.
Terminal opens an external terminal in active Source. **Git** opens an external terminal in the current Git root and immediately runs `git status --short`; it must not duplicate **Open Project**.
- **MIRROR** — current project MIRROR options and commands.
- **SOURCE ACTIONS** — registry/source-container operations: rebuild dropdown, Clone Source, batch preview MIRROR, batch safety, batch verify, batch Git status, combined workspace.
- **PROJECT ACTIONS** — selected tree-object operations: refresh tree, load to Editor, open in VS Code, copy relative/full path.
- **Support** — diagnostics and maintenance: Auth Doctor, BLAKE3, Verify Mirror, Storage, Safety Scan, Clean projects.json.

The **SOURCE** badge under **SOURCE ACTIONS** updates after **Batch Git status** and shows the registry-wide projects / clean / dirty / errors summary.

The **Structure** panel is the tree surface. It has three layer switches:

```
PROJECT -> project.source_path
GIT COPY -> project.projection_path
DOCS -> project.docs_path
```

The selected layer and resolved path are shown in a compact header badge. Folder icons update the selected layer location, and clear removes location overrides. Relative project paths in **config/projects.json** resolve from the Hub Manager root; absolute paths remain valid for external storage.

6. Core Panes

Quick — frequent local Git/file commands in a compact Material icon grid. The command cache does not autocomplete into the command field: selecting cached or pinned commands copies text into the manual command textarea, and Run executes that exact text.

The first **Quick** group is **GIT LOCAL**: **init**, **status**, **root**, **log --oneline** and **reflog**. **user config** lives in the Auth block inside **Remote**, **config list** lives in Git backup/maintenance, and graph history remains in the **History** pane so the local block stays compact and diagnostic.

Git coverage is intentionally broad but not opaque. The UI covers the normal developer lifecycle from **git init** and clone through inspection, staging, commits, tags, remotes, branch switching/creation, stash, integration, history/graph, recovery helpers and maintenance. Branch switching, integration, stash and branch-danger templates are grouped in **Branch**, not duplicated in **Quick**. High-risk or rare workflows remain queued command templates so the operator sees and edits the exact command before execution.

Basket — separate commit/checkpoint preparation pane with long fields.

Branch — branch status, **branch -vv**, **switch**, **switch -c**, tags, compare branches, **merge --no-ff**, **revert**, **cherry-pick**, stash and abort/rebase templates.

Remote — remote configuration, push/pull/fetch and Auth tools. It owns:

- remote form: platform, remote name, login/group, repository, full URL;

- recent-value selects for remote fields, applied only by explicit use buttons;
- `push origin`, `pull --ff-only`, `fetch --all --prune`, `remote -v`;
- `push all remotes`, `apply remotes.json`, `origin push URLs`;
- `Auth Doctor`, GitHub/GitLab SSH probes, `gh auth login`, `glab auth login`, Windows Credentials, GitKraken folder and VS Code.

Push commands use `--follow-tags` by default so annotated checkpoint tags are published. `push all remotes` is implemented in Python through Git remote names, not through a PowerShell-only shell loop.

Editor — Markdown/text editor for `.md`, `.markdown`, `.txt`, `.rst`; icon-only toolbar for load/save/paste/copy/clear/VS Code/expand, CodeMirror when available, textarea fallback if the JS editor fails.

Diff — RedLine-style selected/HEAD changes.

History — selected file/repository history.

Details — stats, JSON and raw payload for the selected object.

BLAKE3 — backend probe for real BLAKE3 vs SHA-256 fallback.

Verify Mirror — read-only Source ↔ Hub Data manifest verification. Manifests are built in memory and the result is written to `logs/<project_id>/`.

Storage — configured roots for Manager, Hub Data, Docs, layout checks, machine-local `Code.exe` picker/test/save controls, Safety Scan summary and raw JSON.

Scan projects — support action for importing many projects from a common parent folder into `projects.json`. The result is shown in Storage/terminal as a JSON report.

Clean projects.json — support action for removing missing Source entries and duplicates from the registry. The result is shown in Storage/terminal as a JSON report.

7. Terminal Dock

Terminal dock must show:

- exact command;
- cwd;
- stdout;

- stderr;
- exit code.

Output is decoded through UTF-8/OEM/Cyrillic fallbacks and rendered as HTML/ANSI so Windows Git and Cyrillic paths remain readable.

8. Git Engine

Base module:

```
system_core/core/git_engine.py
```

The app uses real `git` through `subprocess`. Stable data is parsed from porcelain formats, and raw output remains visible in terminal dock.

Hub Manager-created commits must stage only paths accepted by the active Hub projection profile, plus marker files such as `.gitkeep`.

Normal model:

```
Source/.git
Hub Data/<project>/.git
```

Sidecar Git directories are not used because they make ownership unclear after moves and recovery.

9. Auth Policy

Hub Manager does not store GitHub/GitLab tokens, passwords or private keys.

Authentication is delegated to:

- SSH keys + ssh-agent;
- Git Credential Manager;
- GitHub CLI;
- GitLab CLI;
- VS Code / GitKraken for setup and conflict work.

Auth Doctor should run non-interactive probes and avoid hanging on password prompts.

9.1. Forgejo / Gitea

Self-hosted instances are described in `config/forgejo_hosts.json` (address, SSH user/port, preferred URL type, cached login). The file holds no secrets.

Modules:

<code>system_core/core/forgejo_api.py</code>	API v1: version, user, repos, create
<code>system_core/core/git_credentials.py</code>	git credential fill/approve/reject
<code>system_core/core/forgejo_service.py</code>	sign-in, repositories, Auth Doctor report

The account contract is a personal access token sent as `Authorization: token <TOKEN>`. The token is verified against `/api/v1/user` before it is handed to the external credential helper, and is never written to `config/*.json`. OAuth2/OIDC sign-in is deliberately not used: Forgejo has not implemented OAuth2 scopes, such a token carries administrative rights over the account, is short-lived, and would need a separate mechanism to serve `git push`. See `docs/GIT_AUTH_STRATEGY.md` for the reasoning.

All probes stay non-interactive: `GIT_TERMINAL_PROMPT=0`, and a missing entry is reported as "no token stored" rather than as a failure.

10. Storage Policy

Recommended layers:

<code>Audion_Hub_Manager</code>	independent app/codebase
<code>Audion_Hub_Data</code>	Git-backed technical projections
<code>Audion_Docs</code>	neutral Markdown/text docs folder
<code>Full Projects</code>	source-of-truth projects

Docs can be Obsidian, LogSeq, a VS Code folder, Syncthing/cloud folder or just a directory. Hub Data does not depend on that choice.

11. Testing

Base checks:


```
python -m compileall -q system_core
python -m pytest -q tests
python system_core\ui_nicegui\app.py --smoke
```

Tests should cover:

- profile parsing;
- include/exclude scan;
- MIRROR deletion;
- same-size strict hash compare;
- Source ↔ Hub Data BLAKE3 mirror verification;
- `.gitkeep` creation/removal;
- git status parser;
- terminal decoding;
- safety scan.

12. Documentation Artifacts

Markdown is the source of truth.

PDF and PPTX are release/presentation artifacts. They should not automatically rewrite source documentation and should not enter the Hub commit pathset unless the active profile allows them.

Docs/Obsidian/LogSeq/VS Code docs folders are not BLAKE3 mirror verification targets. Verify Mirror checks Source ↔ Hub Data; Docs remains a readable derived layer, not a separate canonical copy.