TECH SPEC RU — Audion Hub Manager

Содержание

1. Назначение

Audion Hub Manager — локальный commander-интерфейс для связки:

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

Приложение не заменяет VS Code и не привязывает пользователя к Obsidian, LogSeq или другому Markdown-инструменту. Оно делает MIRROR, Git-состояние, diff, commit, safety scan и checkpoint flow видимыми в одном окне.

2. Архитектурный принцип

```
Project first.
Hub follows.
```

Full Project остаётся источником истины. Hub Data — фильтрованная техническая проекция. Docs — необязательная человекочитаемая витрина.

Hub не создаёт документацию из воздуха. Он зеркалит то, что уже есть в проекте: `README`, `TECH_SPEC`, `AGENTS`, `CHANGELOG`, `docs/`, `prompts/`, `reports/` и разрешённый код/конфиги.

2.1. Реестр проектов

`config/projects.json` описывает не один общий Source-контейнер, а набор отдельных проектов. Dropdown `Проект` переключает активную запись целиком:

```
source_path     -> Project layer
projection_path -> Mirror / Hub Data layer
docs_path       -> Docs layer
profile         -> правила MIRROR и commit allowlist
default_branch  -> ветка Git-команд
```

UI-слой `Project` над деревом показывает `source_path` выбранной записи. Это сознательно отличается от общей идеи Source-папки/контейнера, где могут лежать десятки проектов.

`Scan projects` сканирует родительскую папку с множеством проектов, определяет реальные корни по project markers и развитой структуре файлов языков программирования, пропускает launcher-only `.cmd` папки и добавляет недостающие записи в `projects.json`. Hub/Data и Docs ветки, попавшие внутрь scan area, должны пропускаться.

`Clean projects.json` удаляет из реестра только записи с отсутствующим `source_path` и дубликаты. Операция находится в Support, пишет отчёт в Storage/ terminal и никогда не удаляет Source, Hub Data, Docs или другие проектные папки.

3. MIRROR Engine

Базовый модуль:

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

Основной алгоритм:

```
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
```

Правила:

- Source не изменяется MIRROR-движком.
- Projection может удалять stale-файлы.
- `.git/**` защищён.
- Реальный apply требует явного намерения.
- Filtered profile должен иметь непустой include/allowlist.

4. Projection Profiles

Профили живут в:

```
config/projection_profiles.json
```

Dev-Git профиль должен включать:

- исходники;
- Markdown/text документацию;
- lock-файлы и reproducibility manifests;
- CI/build descriptors;
- formatter/linter configs;
- полезные shared `.vscode/tasks.json`, `launch.json`, `extensions.json`.

Профиль должен исключать:

- runtime payload;

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

5. UI Layout

Основная раскладка:

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

Правые вкладки Work Area расположены двумя смысловыми рядами:

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

Каждый заголовок правой вкладки начинается с Material-иконки. Command buttons и заголовки правых вкладок имеют tooltips с задержкой появления 1200 ms и hide/fade 80 ms.

Панели не должны быть свалены в один инспектор. Каждая отвечает за свою рабочую роль.

Левая панель делится на:

- `Open` — открытие текущего Project/Source, Mirror, Docs Folder, VS Code, Terminal и Git. `Terminal` открывает внешний терминал в активном Source. `Git` открывает внешний терминал в текущем Git-root и сразу выполняет `git status --short`; он не должен дублировать `Open Project`.
- `MIRROR` — параметры и команды текущего проекта.
- `SOURCE ACTIONS` — операции над всем реестром/Source-контейнером: rebuild dropdown, Clone Source, batch preview MIRROR, batch safety, batch verify, batch Git status, combined workspace.
- `PROJECT ACTIONS` — операции над текущим выбранным объектом дерева: refresh tree, load to Editor, open in VS Code, copy relative/full path.

- `Support` — диагностика и обслуживание: Auth Doctor, BLAKE3, Verify Mirror, Storage, Safety Scan, Clean projects.json.

`SOURCE`-бейдж под `SOURCE ACTIONS` обновляется после `Batch Git status` и показывает summary всего реестра: projects / clean / dirty / errors.

Панель `Структура` — это поверхность дерева. В ней три переключателя слоя:

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

Выбранный слой и разрешённый путь показываются компактным badge в header. Иконки папок обновляют расположение выбранного слоя, a clear удаляет overrides. Относительные пути в `config/projects.json` считаются от корня Hub Manager; абсолютные пути остаются допустимыми для внешнего хранилища.

6. Core Панели

`Quick` — частые локальные Git/file команды в компактной сетке Material-иконок. Command cache не подставляет autocomplete в поле: выбор cached/pinned команды копирует текст в ручной command textarea, a Run выполняет именно этот текст.

Первый блок `Quick` — `GIT LOCAL`: `init`, `status`, `root`, `log --oneline`, `reflog`. `user config` находится в Auth-блоке внутри `Remote`, `config list` — в Git backup/maintenance, Graph-история остаётся во вкладке `History`, чтобы локальный блок был компактной диагностикой.

Git-покрытие намеренно широкое, но не opaque. UI закрывает обычный lifecycle разработчика от `git init` и clone до inspect, staging, commits, tags, remotes, branch switching/creation, stash, integration, history/graph, recovery helpers и maintenance. Branch switching, integration, stash и branch-danger templates сгруппированы в `Branch`, а не дублируются в `Quick`. Опасные или редкие сценарии остаются queued command templates, чтобы оператор видел и редактировал точную команду перед запуском.

`Basket` — отдельное окно подготовки commit/checkpoint с длинными полями.

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

`Remote` — настройка remotes, push/pull/fetch и Auth tools. В ней находятся:

- remote form: platform, remote name, login/group, repository, full URL;
- recent-value selects для remote fields, которые применяются только явными 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 и VS Code.

Push-команды по умолчанию используют `--follow-tags`, чтобы annotated checkpoint tags публиковались вместе с веткой. `push all remotes` реализован в Python через имена Git remotes, а не через PowerShell-only shell loop.

`Editor` — Markdown/text редактор для `.md`, `.markdown`, `.txt`, `.rst`; icon-only toolbar для load/save/paste/copy/clear/VS Code/expand, CodeMirror при наличии, textarea fallback при сбое JS-редактора.

`Diff` — RedLine-style вывод selected/HEAD изменений.

`History` — история выбранного файла/репозитория.

`Details` — stats, JSON и raw payload выбранного объекта.

`BLAKE3` — backend probe для реального BLAKE3 vs SHA-256 fallback.

`Verify Mirror` — read-only Source ↔ Hub Data manifest verification. Манифесты строятся в памяти, результат пишется в `logs/<project_id>/`.

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

`Scan projects` — support action для импорта множества проектов из общей родительской папки в `projects.json`. Результат отображается в Storage/terminal как JSON-отчёт.

`Clean projects.json` — support action для удаления missing Source entries и дубликатов из реестра. Результат отображается в Storage/terminal как JSON- отчёт.

7. Terminal Dock

Terminal dock должен показывать:

- exact command;

- cwd;
- stdout;
- stderr;
- exit code.

Вывод декодируется через UTF-8/OEM/Cyrillic fallbacks и отображается как HTML/ANSI, чтобы Windows Git и русские пути не превращались в mojibake.

8. Git Engine

Базовый модуль:

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

Используется реальный `git` через `subprocess`. Стабильные данные читаются из porcelain-форматов, raw output остаётся видимым в terminal dock.

Hub Manager-created commits должны stage только те пути, которые допустил бы активный Hub projection profile, плюс marker-файлы вроде `.gitkeep`.

Обычная модель:

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

Sidecar Git directories не используются: они ухудшают прозрачность владения после переносов и восстановления.

9. Auth Policy

Hub Manager не хранит GitHub/GitLab tokens, passwords или private keys.

Аутентификация делегируется:

- SSH keys + ssh-agent;
- Git Credential Manager;
- GitHub CLI;
- GitLab CLI;

- VS Code / GitKraken для setup и conflict work.

Auth Doctor должен запускать non-interactive probes и не зависать на password prompts.

9.1. Forgejo / Gitea

Self-hosted инстансы описываются в `config/forgejo_hosts.json` (адрес, SSH user/port, предпочитаемый вид URL, кэш логина). Секретов файл не содержит.

Модули:

```
system_core/core/forgejo_api.py       API v1: version, user, repos, create
system_core/core/git_credentials.py   git credential fill/approve/reject
system_core/core/forgejo_service.py   вход, репозитории, отчёт для Auth Doctor
```

Контракт учётной записи — personal access token с заголовком `Authorization: token <TOKEN>`. Токен проверяется запросом `/api/v1/user` и только затем передаётся внешнему credential helper; в `config/*.json` он не записывается. OAuth2/OIDC-вход сознательно не используется: в Forgejo OAuth2 scopes ещё не реализованы, такой токен даёт административные права на учётную запись, короткоживущий и требует отдельного механизма для `git push`. Обоснование — `docs/GIT_AUTH_STRATEGY.md`.

Все probes остаются non-interactive: `GIT_TERMINAL_PROMPT=0`, отсутствие сохранённой записи трактуется как «токена нет», а не как ошибка.

10. Storage Policy

Рекомендуемые слои:

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

Docs может быть Obsidian, LogSeq, VS Code папкой, Syncthing/cloud folder или просто каталогом. Hub Data от этого не зависит.

11. Testing

Базовые проверки:

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

Тесты должны покрывать:

- 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 — источник истины.

PDF и PPTX — release/presentation artifacts. Они не должны автоматически переписывать исходную документацию и не должны попадать в Hub commit pathset, если активный профиль их не допускает.

Docs/Obsidian/LogSeq/VS Code docs folders не являются целью BLAKE3 mirror verification. Verify Mirror проверяет Source ↔ Hub Data; Docs остаётся читающим производным слоем, а не отдельной канонической копией.