# Decisions

## 2026-05-04: GUI Shell Over CLI

The GUI layer is a shell over the existing CLI, not a rewrite.

Rationale: Audion tools already have working CMD/FZF workflows. The GUI should make common actions safer and more visible while preserving terminal truth.

## 2026-05-04: Separate Server And Window

Use:

```
launcher_gui.cmd -> system_core/ui_nicegui/window.py -> app.py --no-browser -> pywebview
```

Do not default to NiceGUI `native=True`.

Rationale: the separate process model survived the real canary better, avoids browser surprises, and makes port conflicts easier to diagnose.

## 2026-05-04: Dark Terminal-First Layout

Default layout:

- left: staging, folders, actions, parameters;
- right: status and terminal log;
- terminal log around 2/3 of window height or more.

Rationale: many Audion tools were born as CMD/FZF utilities. Their output is not secondary; it is part of the UX.

## 2026-05-04: Compact Ghost Buttons

Use flat blue text buttons with subtle hover frame instead of large filled buttons.

Rationale: future tools may have 15-20 commands. Fixed large buttons waste vertical space and break with long Russian labels.

## 2026-05-04: Operation Buttons Left-Align Their Labels

Command rows use a dedicated operation-button class. Labels are aligned left inside a fixed command column and overflow only to the right with ellipsis. Short toolbar/folder buttons may stay centered.

Rationale: NiceGUI/Quasar `q-btn` can clip centered long labels from both sides, hiding the beginning of the command. Wider windows do not fix the internal clipping.

## 2026-05-04: Laptop Compression Is A Baseline Requirement

The two-column layout must stay active until about `900px` CSS width and remain usable when compressed toward a WUXGA `1920x1200` laptop at Windows 150% scaling.

Rationale: early wide breakpoints such as `1420px` caused the terminal to fall below the commands. Window size alone does not solve layout resilience.

## 2026-05-07: 1600x900 Is The Roomy Default Window

pywebview windows start at `1600x900` with a practical `1180x720` minimum. Large forms use compact parameter grids, subdued field borders, stable scrollbar gutters, and a draggable splitter between commands and terminal.

Rationale: real Audion Docs AI usage benefits from a spacious first launch, but the GUI must still be resizable and usable on smaller logical workspaces.

## 2026-05-07: Form Order Follows User Decisions

Group related choices together and order fields by the user's decision flow rather than by CLI argument order. Provider key and model belong together; rules, instructions, filters, and rare parameters can sit below. Use radio buttons for small fixed one-of-several choices and searchable selects for long or dynamic lists.

Rationale: Audion Docs AI showed that complex LLM/API tools become understandable when the first visible block answers "what access/model will run this?" and secondary constraints do not interrupt that pair.

## 2026-05-07: One Visible Action Per User Outcome

Do not expose duplicate actions when one command already produces the complete user-facing result. Also avoid duplicate favorite controls: one explicit action or one checkbox, not both for the same list on the same screen.

Rationale: labels such as "audit" vs "full workflow" or checkbox "favorite" vs button "favorite" created more cognitive load than power. CLI/TUI may keep expert wrappers, but the GUI should present the clean user outcome.

## 2026-05-07: Actions Must Name Their Object

Avoid detached, objectless commands. If a screen contains several possible targets, the action label must name the target object or be visually bound to exactly one field: `Favorite model`, `Favorite API key`, `Favorite instruction`, `Check model`, `Delete quick instruction`.

Rationale: Audion Docs AI showed that a lone `Favorite` checkbox or button becomes ambiguous as soon as key, model, task instruction, and quick instruction coexist in one form.

## 2026-05-07: Advanced Fields Collapse By Default

When a form grows beyond the primary decision path, put rare fields such as chunk sizes, retries, timeouts, manual overrides, and strictness toggles into a collapsible advanced block. Persist the block state in GUI settings.

Rationale: Audion Docs AI showed that power settings are useful, but seeing them all the time makes the main action feel further away than it is.

## 2026-05-07: Model List Is Not Model Access

For API projects, keep dynamic model list refresh separate from selected-model smoke checks. The smoke check is explicit, small, cached, and dated.

Rationale: a provider can list historical models that the current account cannot run. A cached selected-model status prevents confusing access errors with code regressions.

Corollary: avoid hand-curated fixed model ids in project YAML or sidecar list files when a live provider list/cache exists. A developed LLM model selector with cache, favorites, and selected-model smoke status replaces old files such as `models.yaml`, `models.txt`, or static provider profile lists. Static config should hold stable provider settings, env names, prompts, and runtime limits; model ids belong to the GUI cache/favorites/smoke layer.

## 2026-05-07: Visual Smoke Screenshots Are Part Of Porting

After layout changes, save screenshots of the root screen, a representative command form, any changed advanced state, and the terminal after a short successful run.

Rationale: GUI regressions are often visual and contextual. A screenshot catches crowding, duplicated controls, clipped labels, harsh borders, and missing final status faster than code review alone.

## 2026-05-04: Stage External Input Locally (superseded by 2026-07-16)

GUI should offer `Add files...` and `Add folder...` actions that copy selected items into `input`.

Rationale: safer than long operations directly on network paths, removable drives, or deep user folders.

## 2026-07-16: Use The Canonical Direct-Route Workbench

The current fleet standard supersedes the staging toolbar. Use byte-identical `workbench.py` modules with exact RU labels `Источник`, `Добавить файл...`, `Назначение`, `Сбросить`, `Удалить`, `Список` and EN labels `Source`, `Add file...`, `Target`, `Reset`, `Delete`, `List`. Selected routes become active service paths; staging is a separate project operation only when the backend explicitly requires it.

## 2026-05-04: Language And Theme Are Conservative

Dark mode is the default. Runtime language switching may reload the UI. Light theme and full international edition are later refinements.

Rationale: live switching caused instability during the canary. Reliability wins.

## 2026-05-06: Completion State Is Persistent

The right terminal panel keeps a small final-status indicator below the log: idle grey, running blue, success green, failure red.

Rationale: transient notifications can be missed when the GUI window is inactive. The terminal footer is a permanent, low-noise place to show that the last operation really finished.

## 2026-05-20: Notifications Must Not Depend On Deleted Slots

Long-running operations may outlive the child screen or button that launched them. Completion/failure notifications should be delivered through live NiceGUI clients (`app.clients()` or the template `safe_notify`) instead of the current handler slot after a long `await`.

Rationale: users naturally browse other menus while a batch runs. If NiceGUI deletes the original slot, `ui.notify()` from that stale context can turn a successful run into `RuntimeError: The parent element this slot belongs to has been deleted.` The durable result remains `state`, terminal output, logs and the terminal footer; toast is helpful but non-critical.

Corollary: opening source, target, `LOGS`, `CONFIG`, `REPORT`, or `TOOLS` should not show toast notifications. The OS file manager is the confirmation. Picking a new route, deleting content, and imports may use toasts because they change project state.

## 2026-05-06: Hide Windows CLI Helpers At Process Creation

Windows subprocess helpers, especially `pwsh.exe` and `powershell.exe`, must be hidden through Python process creation flags (`STARTUPINFO/SW_HIDE` and `CREATE_NO_WINDOW`). `-WindowStyle Hidden` may stay as a second layer, but it is not sufficient by itself.

Rationale: PowerShell can briefly create a console window before its own flags take effect. In batch tools this creates distracting per-file flashes. The GUI terminal should mirror output; separate CLI mirror windows should not be the default UX.

## 2026-05-06: CMD Encoding Is A Build Gate

All project-owned `.cmd` files are UTF-8 without BOM and strict CRLF. The template provides `install\Check-CmdEncoding.cmd -Fix` / `install\Repair-CmdEncoding.ps1 -Fix` as an

explicit repair step, while offline install, verify, doctor, tests, and release packaging use check-only gates.

Portable runtime build must not self-repair CMD files while it is running. Rewriting the active `.cmd` launcher can make `cmd.exe` parse stale arguments as commands, for example `-Fix is not recognized`.

Rationale: CMD encoding and line endings are too easy to break during GUI porting. The standard should live in the template tooling, not in repeated verbal reminders.

## 2026-05-09: SH LF Is A Build Gate

All project-owned `.sh` files are UTF-8 without BOM and strict LF. The template provides `system_core\core\sh_lf.py --fix`, and `doctor.py` reports SH line-ending failures next to CMD encoding failures.

Rationale: WSL/bootstrap work made it clear that Linux scripts are just as fragile in the opposite direction as CMD files. The template must protect both Windows and Linux launch surfaces.

## 2026-05-09: System Output Uses Byte Streaming

`run_process()` streams child process output as bytes and decodes with UTF-16-ish, UTF-8, OEM, locale, `mbcs`, `cp866` and `cp1251` fallbacks. It must not be simplified back to `text=True, encoding="utf-8"`.

Rationale: `wsl`, `netsh`, `reagentc`, `cmd.exe` and Windows PowerShell can emit different encodings, especially on Russian Windows. A GUI terminal with unreadable output is not trustworthy.

## 2026-05-21: GUI Terminal Renders ANSI Safely

The GUI terminal renders subprocess output through `system_core/core/ansi.py`: normal text is escaped first, then whitelisted ANSI SGR styles become HTML spans. The widget is an HTML terminal surface, not a textarea-only log. File logs stay plain UTF-8 and strip ANSI before writing.

Rationale: Doctor, installers and CLI helpers often use ANSI status colors. Showing raw `\x1b[36m` sequences is noisy, while blindly removing ANSI loses useful status cues. Safe ANSI-to-HTML keeps Cyrillic, color and readable disk logs together.

## 2026-05-09: WSL Is A Unified Module Pattern

Future WSL GUI work should ask for distro name, image file and install folder directly. Disk-specific `WSL_E` / `WSL_S` style choices can remain legacy scan locations but should not be the primary model.

Rationale: WSL2 supports explicit install locations and local `.wsl` images. The GUI should model the user's real decision instead of preserving old disk blocks.

## 2026-05-09: Admin UX Beats Access-Denied Logs

Known admin-only operations should use an elevated launcher or per-operation UAC handoff that replays logs into the GUI terminal. Read-only/status operations should stay usable without elevation.

Rationale: access-denied spam is not a workflow. System tools should either run with the rights they need or tell the user upfront what is missing.

## 2026-08-12: Root Is A Switcher, Not A List Of Doors

The root command screen is a row of tabs, one per top-level group, with that tab's commands underneath. A command carrying fields unfolds where it stands - its own parameters and a run button named after the action.

Rationale: these programs have a handful of commands. Walking into a child window to press an identical `Run` was a step that carried no information, and the root list of section titles told the user nothing they could act on. Service operations sit above the tabs because they belong to the program rather than to any one tab.

## 2026-08-12: Choices Are Buttons

Radio groups and the browser selection are rows of switch buttons. The chosen one is washed with a translucent Quasar blue, the rest stay outlined - browsers in their own brand colour. Amber remains reserved for run buttons.

Rationale: a column of dots of unequal width is read item by item; a row of buttons is read at a glance. Three states - run it, chosen, not chosen - now look different from each other, which they did not while everything was a chip.

The buttons are created with `color=None`. Quasar's stylesheet lives in a cascade layer, and an `!important` inside a layer outranks an `!important` outside it at any specificity, so `bg-primary`

cannot be overridden from the project CSS - it has to be prevented.

## 2026-08-12: Short Captions, Long Tooltips

Labels, row descriptions and hints stay at 7-10 words, often one or two. The explanation - what happens, what a wrong answer costs - lives in `tooltip` / `tooltip_ru`, which may be as long as the subject needs.

Rationale: the window is a control panel, not a manual. Text beside a control is scanned, text under the pointer is read on purpose.

## 2026-08-12: Updating Happens Where The Build Lies

`Update` replaces `App` in the folder Source points at and never publishes a second copy into the Target folder. The Target folder is for new builds.

Rationale: a user who pointed the picker at a build on a flash drive and then found it unchanged there had every right to be confused. The swap goes through a rename so a running browser cannot leave a half-replaced build behind.