Answers Become Interfaces

WorkPage OS, typed plans, safe static rendering, and evidence-bearing generated operating surfaces

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

Answers Become Interfaces

Abstract

Chat interfaces are excellent for discovering intent and poor at carrying serious work to completion. A long answer may contain a plan, commands, decisions, evidence requirements, and checkpoints, yet the user must manually convert the prose into an operating environment.

WorkPage OS turns intent and source material into a typed `WorkpagePlan`, then renders a safe, static, portable, interactive **Workpage**. A Workpage may be a build runbook, learning surface, troubleshooting tree, decision page, research dossier, creative production board, or completion audit. It includes copy controls, checkboxes, progress, evidence fields, branching gates, notes, and export. Basic use requires no server, account, or model after generation.

The model does not generate arbitrary final HTML on the main path. It proposes a structured plan. A deterministic renderer produces the interface. Projects, plans, revisions, sources, exports, critic reports, and evidence remain durable artifacts. Completion claims require evidence.

The existing v1.1 specification and nineteen-phase runbook define a build-ready local-first Tauri 2 application with Rust, React/TypeScript, SQLite, fixture generation, Ollama and LM Studio adapters, preview isolation, security gates, packaging, and an independent completion audit.

1. Chat is the prompt; the Workpage is the workplace

A user can receive a technically correct answer and still fail to perform the task because:

- commands are buried in prose;
- progress is not persistent;
- decision points are unclear;
- evidence is not collected;
- the answer is hard to resume;
- steps cannot be checked;
- failure branches are missing;
- the model announces completion before the user acts.

WorkPage OS externalizes the answer into a task-specific interface.

The core thesis is:

> Chat is for discovering intent. Workpages are for doing the work.

2. Workpage types

Build Runbook

Phased implementation with copy-paste commands, PASS/FAIL gates, evidence, stop lines, and final audit.

Learning Surface

Objectives, explanations, examples, exercises, quiz, teach-back, source links, and mastery gates.

Troubleshooting Guide

Observed symptoms, diagnostic commands, result fields, branching hypotheses, repairs, logs, and final verification.

Decision Surface

Options, constraints, evidence, tradeoffs, assumptions, consequences, and accepted decision.

Research Dossier

Claims, sources, conflicts, evidence status, notes, and revision.

Creative Production Board

References, locks, shot/asset plan, costs, reviews, and accepted outputs.

The v1 architecture can support these archetypes through one plan and card registry.

3. Structured generation before HTML

The model emits a `WorkpagePlan`:

```
WorkpagePlan {
  title
  purpose
  page_type
  source_manifest
  sections[]
  cards[]
  gates[]
  evidence_requirements[]
  actions[]
  export_policy
  completion_contract
}
```

Card types have schemas. The validator rejects missing or unsafe structures. The renderer owns HTML, CSS, accessibility, and runtime behavior.

This avoids arbitrary model-generated script and inconsistent interface code.

4. Static-first export

An exported Workpage is standalone HTML.

It can support:

- copy buttons;
- checkboxes;
- short notes;
- local progress;
- reset;
- state export/import;
- print;
- accessible navigation.

It loads no remote scripts or fonts by default. State remains small. Large data and secrets do not enter the export.

A Workpage can be emailed, hosted, zipped, archived, or opened offline.

5. Artifact-first persistence

The system saves:

- source prompt;
- source files;
- WorkpagePlan;
- rendered HTML;
- revisions;
- critic reports;
- evidence;
- exports;
- completion state.

Everything serious becomes an artifact. A provider response is not the only record.

The local application uses SQLite for projects and revisions, with files for large exports/assets where appropriate.

6. Provider boundary

The Rust side owns:

- network;
- provider adapters;
- secrets;
- filesystem;
- database commands;
- export;
- security validation.

The renderer receives provider status and generated plans, never API keys.

Initial adapters include:

- fixture provider;
- Ollama;
- LM Studio;
- optional OpenAI-compatible provider after secrets pass.

Each adapter maps its own structured-output behavior. “OpenAI-compatible” is not assumed to mean identical.

7. Preview isolation

The application preview is more privileged than the exported page. They must be separate runtimes.

Preview-only actions may ask the host app to:

- save revision;
- open source;
- attach evidence;
- regenerate block;

- export.

Export runtime actions remain local and non-networked.

Generated shell commands are never automatically executed in v1.

8. Evidence rails

Workpages distinguish:

- **BLOCKER:** prevents valid completion;
- **PREFERENCE:** affects quality but not validity;
- **CONFIRM:** requires current environment or user verification.

A build phase ends in PASS or FAIL with evidence.

The final completion audit independently rechecks prior phase claims. Earlier “PASS” text is not trusted without the evidence paths.

9. WorkpagePlan revision

A model or user can propose a revision:

```
base_plan_version
operations[]
reason
source_refs
critic_findings
```

The application validates and creates a new immutable revision. Accepted exports remain available.

A small SurfaceDelta-style operation can modify one card instead of regenerating the whole page.

10. Security rules

- no secrets in renderer, SQLite, logs, exports, or generated pages;
- no remote scripts/fonts in standalone output;
- no automatic shell execution;
- narrow Tauri capabilities;
- dialog-origin filesystem handles;
- sandboxed preview;
- Content Security Policy;
- schema limits;
- HTML escaping;
- no unsafe arbitrary plugin access;
- local providers fail clearly when unavailable.

11. Accessibility

The generated surface must have:

- semantic headings;
- keyboard navigation;
- visible focus;
- large targets;
- contrast;

- reduced motion;
- screen-reader labels;
- non-color state;
- printable structure;
- responsive mobile behavior.

A beautiful workpage that cannot be operated by keyboard is not complete.

12. Build runbook as the flagship proof

The v1.1 runbook contains nineteen phases:

0. prime and guardrails;
1. scaffold;
2. design shell;
3. SQLite and types;
4. schemas and cards;
5. static renderer;
6. preview security;
7. fixtures;
8. local adapters;
9. optional cloud;
10. workbench;
11. library/wizard/settings;
12. build runbook generator;
13. learning/troubleshooting;
14. decision/dossier/creative;
15. revision/evidence/cost;
16. tests/accessibility/security;
17. packaging;
18. completion audit.

The runbook itself is a functioning static Workpage with copy controls and local progress. It demonstrates the product form before the application is complete.

13. Completion audit

The final audit verifies:

- Tauri app launches;
- migrations and CRUD survive restart;
- plans validate;
- invalid fixtures fail clearly;
- all card types render;
- exports run standalone;
- copy/check/progress/reset/state export work;
- fixture provider generates all page types;
- adapters work or show honest unavailable state;
- secrets do not leak;
- preview isolation works;
- revision and evidence rails work;
- tests, accessibility, and security pass;
- package artifact launches;
- completion report contains evidence.

Only the audit can declare v1 complete.

14. Relationship to SurfaceMind and Glyph Desk

SurfaceMind provides a general semantic surface and delta substrate. WorkPage OS is a product expression optimized for generated static task interfaces.

Glyph Desk is the immediate Chat–Plan–Make operator shell. Workpages can be artifacts created in Make and governed through Plan.

The relationship is:

```
Chat discovers intent
Plan declares constraints and authority
Make generates a Workpage
Workpage operates the task
Evidence returns to Project Core
```

15. Demonstration

The public Glyphd Labs demo can generate a Workpage from a fixture intent:

`“Prepare a safe, evidence-backed runbook for publishing a paper.”`

The deterministic fixture produces:

- plan;
- sections;
- copyable commands;
- checklist;
- evidence fields;
- completion gate;
- standalone HTML;
- exported state.

An optional local model can later generate a plan, but the fixture path is sufficient for acceptance.

16. Evaluation

Compare prose answer versus Workpage on:

- time to first action;
- task completion;
- step omission;
- evidence quality;
- resume after interruption;
- error recovery;
- user confidence;
- export portability;
- accessibility;
- correction/revision count.

A Workpage should be judged by performed work, not how impressive the generated page looks.

17. Evidence state

The v1.1 corpus is unusually complete as a build specification:

- source truth and verified-anchor policy;

- pressure test;
- complete product spec;
- security boundaries;
- nineteen-phase interactive runbook;
- detailed completion audit;
- build-first instructions.

This is strong `SPECIFIED` and partially `IMPLEMENTED` evidence for the runbook artifact. A fully built packaged WorkPage OS application must be confirmed from the current repository or built under the runbook; this paper does not promote it without a receipt.

18. Limitations

Generated pages can become templated and repetitive. Structured plans may fail to capture unusual workflows. LocalStorage is limited. Standalone HTML cannot safely perform all tool actions. Tauri packaging varies by platform. Local models may produce invalid plans. Static interfaces can become stale.

The architecture deliberately keeps automatic consequential actions out of v1.

19. Conclusion

WorkPage OS argues that the natural output of serious AI assistance is often not another answer.

It is a small operating surface that preserves the answer's structure, helps the user do the work, collects evidence, survives interruption, and can be carried anywhere.

The sentence the product must prove is:

> **Answers become interfaces, and serious tasks become generated operating surfaces.**

Architecture illustration briefs

- **P20-F01 — Answer to Workpage**: Intent and sources → typed WorkpagePlan → validator → deterministic renderer → standalone interactive page → evidence/receipts.
- **P20-F02 — Preview versus export boundary**: Privileged app preview bridge separated from static non-networked export runtime.
- **P20-F03 — Nineteen-phase build**: Phase ladder ending in an independent audit rather than self-reported completion.
- **P20-F04 — Workpage archetypes**: Build, learn, troubleshoot, decide, research, create around one plan/card substrate.

Source register

- **P20-S01** — Workpage OS Spec Pack v1.1 (2026-06-18). `Workpage_OS_Spec_Pack_v1_1.md`
- **P20-S02** — Workpage OS Pressure Test v1.1 (2026-06-19). `Workpage_OS_Pressure_Test_v1_1.md`
- **P20-S03** — Workpage OS Build Runbook v1.1 (2026-06-19). `Workpage_OS_Build_Runbook_v1_1.html`
- **P20-S04** — Workpage OS Build Pack README (2026-06-19). `README_BUILD_FIRST.md`
- **P20-S05** — Glyph Desk Master Specification (2026-07-12). `GLYPH-DESK-MASTER-SPEC.v1.md`

Public-disclosure and IP boundary

This edition publishes the public-safe product architecture and build/audit doctrine. It excludes private provider keys, user projects, proprietary prompts, and patent-claim drafting. It does not claim the packaged desktop application is complete without a current build and completion receipt.