RishFlow — Project Overview & Publication Guide

Version: 2.0 Date: 2026-02-02

---

1. Executive Summary

RishFlow is a desktop file-organizer that helps users tidy files into meaningful folders using rule-based and AI-powered techniques. This document describes the system architecture, features, installation and publishing checklist, AI design choices (including a privacy-first local index and upgrade path to RAG), and clear next steps for preparing a polished public release.

---

2. Key Features

- Modern HTML dashboard (embedded in Python webview) with a clean, themeable interface.
- File organization modes: File Extension, Date Modified, Size Category, AI-based classification (content-aware via `AISmartSorter`).
- Folder stats (file counts and aggregated sizes) and top largest files.
- Activity logging (SQLite) and undo (revert) support.
- Duplicate detection using exact MD5 hashing and perceptual image hashing.
- Lightweight, privacy-first AI index for `.txt` and `.pdf` files plus a simple local search (prototype for RAG).

---

3. System Architecture (brief)

- Frontend: `stitch_rishflow_dashboard_home (1)/code.html` — Tailwind/Tailwind CDN, JS controls, theme persistence in `localStorage`, interacts with Python via `pywebview` JS API.
- Backend: `app.py` — `RishFlowAPI` class exposes methods to JS (`scan_source`, `get_folder_stats`, `start_organizing`, `revert_last`, `index_for_ai`, `query_ai`, `find_duplicates`).
- AI: `ai_sorter.py` — OpenCV + Tesseract heuristics for image/document classification.

- Utilities: `duplicate_finder.py` (hashing), `bulk_rename.py` (renaming strategies).

---

4. Installation & Quick Start

Requirements (recommended): Python 3.11+ (3.13 tested), Windows 10/11.

1. Clone the repo.

2. Create a virtual environment (recommended):

```
python -m venv .venv
.\.venv\Scripts\activate
```

1. Install dependencies:

```
pip install -r requirements.txt
```

1. (Optional) For PDF indexing: `pip install pypdf`

2. Start the app:

```
python app.py
```

1. Browse: Click *Browse* to select a Source folder and see live **Items** and **Size** stats. Use **AI Search** to query `.txt` and `.pdf` content locally.

5. Detailed Feature Explanations

5.1 Folder Stats

- API: `get_folder_stats(folder_path)` returns `total_files`, `total_size` (bytes), `count_by_type`, `size_by_type`, `largest_files` (top 10).
- Frontend updates metrics after folder selection and when organization/revert completes.

5.2 AI Search (Local Prototype)

- `index_for_ai(folder_path)` extracts text from `.txt` and `.pdf` files and keeps a lightweight in-memory index.
- `query_ai(folder_path, query)` performs substring search over the indexed texts and filenames and returns snippets.
- Privacy-first by design: no data leaves the machine unless you later integrate cloud LLMs.

5.3 RAG/LLM Upgrade Path

- For production RAG, add:
- Embeddings (e.g., `sentence-transformers` or OpenAI embeddings)
- Vector store (Chroma or FAISS)
- LangChain for orchestration
- LLM provider: OpenAI (cloud) or a local LLM (e.g., via Ollama, or Llama.cpp)
- Architecture: extract documents → split → embed → store → query → re-rank → use LLM to synthesize answers.

6. Privacy & Security Considerations

- The current AI index is local and stores only text snippets in memory. No network calls are made by default.
- If you enable cloud LLMs (OpenAI), be explicit in UI and docs that files (or embeddings) will be sent to a third-party provider.
- Sanitize user inputs, limit file types for AI indexing, and provide clear opt-in prompts for cloud features.

7. UI & Theme Guidance

- The UI uses Tailwind and CSS variables / gradient overlays for themes (`ocean`, `deep`, `sunset`).
- To add new themes, add style blocks in `code.html` and a corresponding button that calls `setTheme('yourtheme')`.
- Persist theme using `localStorage` (already implemented).

8. Tests & CI Recommendations

- Add unit tests for:
- `get_folder_stats` (counts, sizes, edge cases)
- `query_ai` and `index_for_ai` (with sample files)
- Add GitHub Actions to run tests on push, linting (flake8/ruff), and to build a release artifact (optional: PyInstaller build via `build.py`).

9. Packaging & Publishing Checklist (What you need to do)

1. Finalize UI copy and screenshots.
2. Add detailed README and usage samples.
3. Add automated tests and CI.
4. Choose an LLM strategy (cloud vs local). If cloud, ensure user consent and API key configuration is clear.
5. Update `build.py` and `requirements.txt` with any added dependencies.
6. Create release notes and a marketing one-pager.
7. Run the PyInstaller build and test the generated executable on a clean Windows environment.
8. (Optional) Create an installer using `Inno Setup` or similar.

10. Suggested Release Assets (I'll create these if you want)

- Short demo video (2–3 minutes) showcasing key flows.
- `docs/RishFlow_Docs.pdf` — this publication guide in PDF.
- `docs/QuickStart.md` — condensed quick start for README.
- Screenshots and a one-page slide deck for product listing.

11. Contact & Next Steps

If you'd like, I can: - Produce the 2–3 minute MP4 demo (slides + synthetic narration), - Generate `docs/RishFlow_Docs.pdf` and place it in the `docs/` folder (I will do that now), and - Create a `Publish.md` with step-by-step build & release commands.

Please confirm if you want the video included and which narration option: **A: synthetic narration (I produce MP4)** or **B: recording script for you to record**.

End of document.