ACCESSIBILITY UTILITY

Intelligrit LLC | Open Source Product Case Study

Build the utility you need, then make it fast enough to disappear.

FFWF turned a real accessibility and workflow problem on macOS into a low-drag menu bar utility with fuzzy search, keyboard-first navigation, and event-driven performance.

39	1	v1.2	MIT
COMMITS	PRIMARY BUILDER	RELEASED	LICENSED

The Short Version

FFWF exists because the built-in macOS path for finding the right window is too slow, too linear, and especially frustrating for blind and low-vision users relying on VoiceOver. Intelligrit built a menu bar utility that makes window switching searchable, keyboard-first, and fast enough to feel invisible.

The product is small on purpose: a focused local utility, built in Swift and SwiftUI, with a practical architecture around Accessibility APIs, fuzzy matching, hotkey capture, and event-driven refresh. It ships as a simple app bundle, supports Homebrew installation, and stays out of the user's way until needed.

> *FFWF is not interesting because it is a menu bar app. It is interesting because it shows how lived accessibility pain, practical engineering judgment, and*

low-drag shipping can produce software people actually keep using.

The Problem

macOS gives users tools for moving between windows, but not a good way to search for the specific one they want. The built-in switcher is effectively linear navigation. That is annoying for anyone working across many

windows and dramatically worse for VoiceOver users, where visual scanning shortcuts do not help and repeated traversal adds real friction.

That is exactly the kind of problem Intelligrit cares about: small, painful, recurring workflow drag that looks minor until you experience it every day.

- No direct searchable window targeting.
- Keyboard-first workflows break down when the list gets long.
- Accessibility support exists, but the default path is still high-friction.
- Utility software has to feel instantaneous or people abandon it.

What Shipped

A local-first macOS utility for fast window targeting, fuzzy search, and accessible keyboard-only navigation.

What FFWF Actually Does

- **Global hotkey activation** with configurable keyboard shortcuts.
- **Fuzzy search over open windows** using window titles and app names.
- **Multi-term matching** for faster narrowing in crowded environments.
- **VoiceOver announcements** for result counts and selected windows.
- **Search history and autocomplete** for repeated workflows.
- **Low-overhead refresh behavior** so the app is active only when it needs to be.

Architecture

The internal structure is simple and intentional.

Component	Role
`WindowManager`	Enumerates windows through the macOS Accessibility API and keeps the current window set fresh.
`FuzzyMatcher`	Scores matches with exact-match, prefix, word-boundary, and position-aware logic.
`ContentView`	Owns the keyboard-first interaction model, focused search flow, result limiting, and VoiceOver announcements.
`FFWFApp`	Handles menu bar behavior, global hotkey registration, permissions flow, and floating search window management.
`SearchHistory`	Stores recent searches and supports repetitive workflows without extra typing.

Performance Decisions

FFWF is a good example of Intelligrit's engineering taste: not high ceremony, not overbuilt, just the right optimizations in the right places.

- **Concurrent window enumeration** using parallel processing across running applications.
- **Pre-lowercased strings** so matching does not pay repeated normalization costs.
- **Early exits in matching** so non-matches fail fast.
- **Icon caching** so expensive lookups are not repeated per row.
- **Refresh while visible, not always** so the app stays cheap when hidden.
- **Stable IDs and focused UI updates** so the interaction stays responsive.

```
// The shape of the system is simple:
// enumerate windows -> score matches -> announce selection -> activate target
// no unnecessary service layer, no remote backend, no platform theater
```

Why It Fits Intelligrit

FFWF is not the main buyer proof. It is proof of accessibility-grounded product judgment, practical shipping, and low-drag system design.

Why This Matters To Buyers

FFWF is not a federal internal web app, but it does show a different part of the company clearly:

- **Accessibility is lived, not decorative.** The problem choice came directly from real workflow pain.
- **Useful software can be small and sharp.** Not everything needs a platform.
- **Local-first software is often the right answer.** No cloud dependency, no phoning home, minimal ops burden.
- **Engineering judgment matters.** The stack and architecture served the workflow instead of marketing the stack.

Technology Positioning

FFWF was built in Swift because it is a native macOS utility and Swift is the right tool in that environment. That does not make Swift a core public Intelligrit stack signal. It makes FFWF proof that Intelligrit chooses technology to fit the problem rather than to satisfy ideology.

The broader Intelligrit lesson is the same one visible in the company's web and backend work: keep the architecture small, keep deployment practical, and use the minimum system that solves the real pain.

What This Proves

- Intelligrit can turn lived operational pain into useful software quickly.
- Accessibility and usability are part of the build logic, not just compliance add-ons.
- The company can ship polished utility software with packaging, release flow, and distribution hygiene.

Small focused tools can still demonstrate strong product judgment and durable engineering taste.

Distribution And Operations

FFWF also shows that Intelligrit finishes the boring but important parts: app bundle creation, code signing support, DMG generation, GitHub release flow, and Homebrew cask distribution. Shipping is not just code completion. It is getting useful software into a form people can install and trust.