Desktop Cleanup App - Design Document

Project Type: Personal productivity tool for macOS
Status: Concept & UI Design Phase
Build Approach: Build for personal use first, then potentially open to others
Date Created: November 10, 2025

Table of Contents

1. Problem Statement
2. Core Concept
3. User Requirements
4. Feature Overview
5. MVP Scope
6. UI Prototype
7. Development Approach
8. Technical Considerations
9. Business Model
10. Next Steps

Problem Statement

The Challenge:
Files constantly pile up on Desktop and Downloads folder due to rushed saving habits. Figuring out the right organizational structure is part of the problem, not just executing it.

User Needs:

- Periodic restructuring to feel organized
- Help with both creating AND maintaining folder structures
- Balance of automation and user control
- Learning system that adapts to personal patterns
- Flexibility in cleanup frequency (background nudges OR scheduled sessions)

Core Concept

"An Intelligent Desktop Butler"

A personal organizer that learns your chaos patterns and helps you build order from them. Think of it as a helpful assistant that:

- Watches your messy folders without judgment
- Suggests intelligent organization based on patterns
- Learns from your decisions over time
- Gives you control over automation level
- Makes cleanup feel quick and satisfying, not overwhelming

User Requirements

File Types

All types pile up:

- Documents (PDFs, Word files, etc.)
- Images (screenshots, downloads)
- Design files (.sketch, .fig, .psd)
- Code files
- Archives (.zip, .dmg)
- Media files

Automation Preferences

- **Option for automatic filing** with high confidence
- **Option for suggested actions** requiring approval
- User decides which mode per rule or globally

Scheduling Preferences

User choice between:

- Background monitoring with gentle nudges
- Scheduled cleanup sessions (monthly deep cleans recommended)

- Threshold-based triggers ("100+ files on desktop")

Learning Approach

- **Pattern learning:** AI observes and learns from user behavior over time
- **Rule-based:** User can define explicit rules upfront
- **Hybrid preferred:** Combination of both approaches

Smart Features (All Yes)

- ✅ AI-powered suggestions
- ✅ Duplicate detection
- ✅ Archiving old/unused files
- ✅ Pattern recognition
- ✅ File never-opened identification

Feature Overview

The Experience

First Launch: Discovery Phase

1. **Initial Scan**
   - Scans Desktop, Downloads, Documents, and designated "collection zones"
   - Shows visual breakdown:
     - Heatmap of file types (% documents vs. images vs. code)
     - Age distribution (today vs. last week vs. 6+ months)
     - Preliminary pattern suggestions
2. **Onboarding Paths**
   - **"Suggest for me":** AI proposes folder hierarchies based on file analysis
     - Example: Work/Projects/ClientName, Personal/Finance/2024, Creative/Photography
   - **"I'll teach you":** User defines rules manually
     - Example: "All .sketch files → Design/Working Files"

Day-to-Day: Background Watcher

- **Menu bar presence:** Clean icon with subtle badge count
- **Gentle nudges:**

- **Gentle nudges:**
  - "15 files on your desktop"
  - "Haven't organized Downloads in 2 weeks"
  - Smart timing: Never interrupts during active work hours
- **Quick Actions from menu bar:**
  - Quick Sort (auto-files confident matches)
  - Review Mode (approve/adjust suggestions)
  - Snooze until [date]

Monthly Deep Clean: Guided Session

- **Calendar notification:** "Ready for your November cleanup? 47 items need attention"
- **Tinder-like review interface:**
  - One file at a time with preview
  - Suggested destination with confidence indicator
  - Swipe/hotkey actions:
    - → Accept suggestion
    - ← Choose different location
    - ↓ Archive (timestamped)
    - ↑ Keep on desktop (mark as intentional)
- **Batch operations:** "These 12 files look similar - apply rule to all?"

Smart Features Breakdown

1. AI-Powered Categorization

- **Content analysis:** OCR on PDFs/images for receipts, invoices, contracts
- **Filename parsing:** Extract client names, project names, version numbers
  - Example: "ClientName_ProjectBrief_v3.pdf" → client, project, version
- **Metadata usage:** Creation date, originating app, tags
- **Pattern recognition:** "You always move Bank of America PDFs to Finance/Statements - automate?"

2. Learning Your Behavior

- Tracks every manual organization action

- Builds personal model over time
- **Confidence scoring:** Only auto-files when 90%+ confident
- **Feedback loop:** Learns from overridden suggestions

3. Duplicate Detection

- Visual similarity for images
- Hash matching for exact duplicates
- Version detection (file_v1, file_v2, file_final_FINAL)
- Side-by-side comparison interface
- Smart suggestions: "Keep newest? Delete all but one?"

4. Archive Intelligence

- Identifies files not opened in 6+ months
- Creates dated archives (Archive/2024-Q4)
- Maintains searchable archive index
- Prevents "lost forever" syndrome

5. Smart Search

- Context-aware file location memory
- Example: "I moved that Q2 Report to Work/Reports/2024/Q2 - want me to open it?"

MVP Scope (Version 0.1)

Build for yourself first. Ruthlessly scoped to actual immediate needs.

Core Features Only:

1. ✅ **Manual scan trigger** (no background monitoring yet)
2. ✅ **Desktop + Downloads only** (primary dumping grounds)
3. ✅ **Rule-based sorting** (no AI initially - define rules manually)
4. ✅ **Review interface** (approve/reject before moving)
5. ✅ **Simple folder structure** (pre-defined by user)

What's NOT in MVP:

- ❌ AI/Machine learning
- ❌ Background file monitoring
- ❌ Duplicate detection
- ❌ Archive management
- ❌ Pattern learning
- ❌ Multiple folder support beyond Desktop/Downloads

MVP User Flow:

1. Click menu bar "Scan Now"
2. App shows all files from Desktop/Downloads
3. Pre-set rules suggest destinations
4. Review each suggestion
5. Approve batch moves
6. Done

Timeline Estimate: 3-6 months for solid MVP (learning Swift as you go)

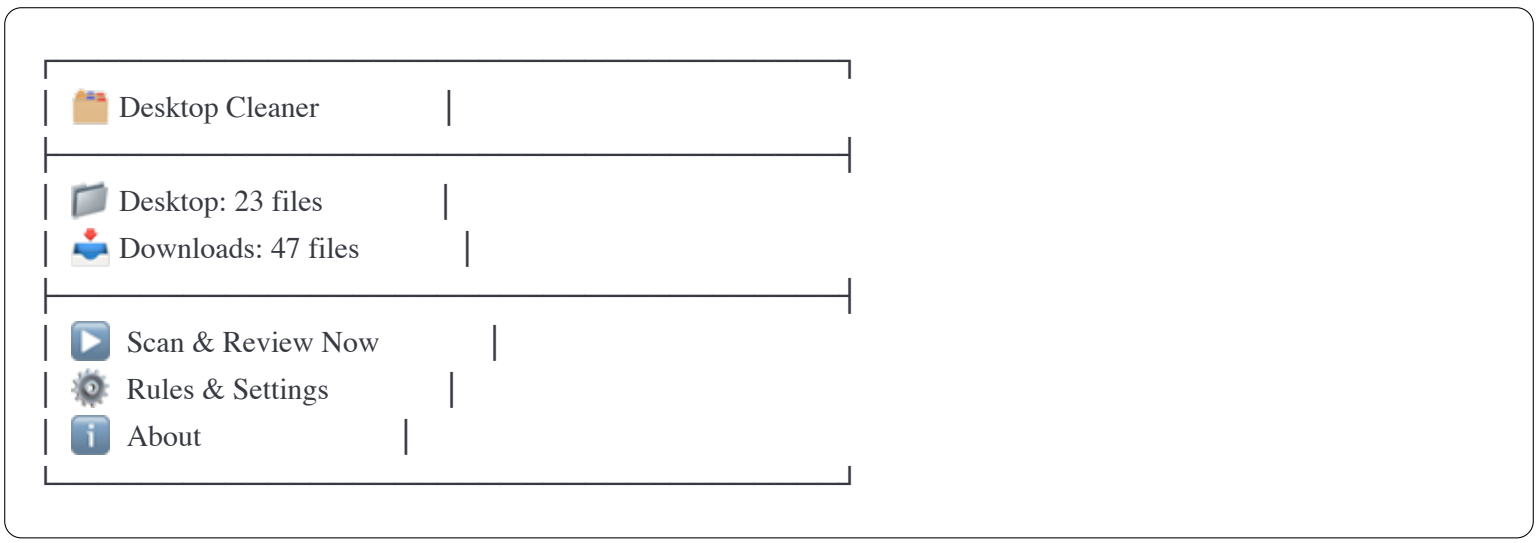
UI Prototype

Three Core Screens

Screen 1: Menu Bar Dropdown

Purpose: Quick status check + launch point

Visual Structure:



Specifications:

- Width: ~250px
- Clean, minimal design
- File counts update on dropdown open
- "Scan & Review Now" opens main window

States:

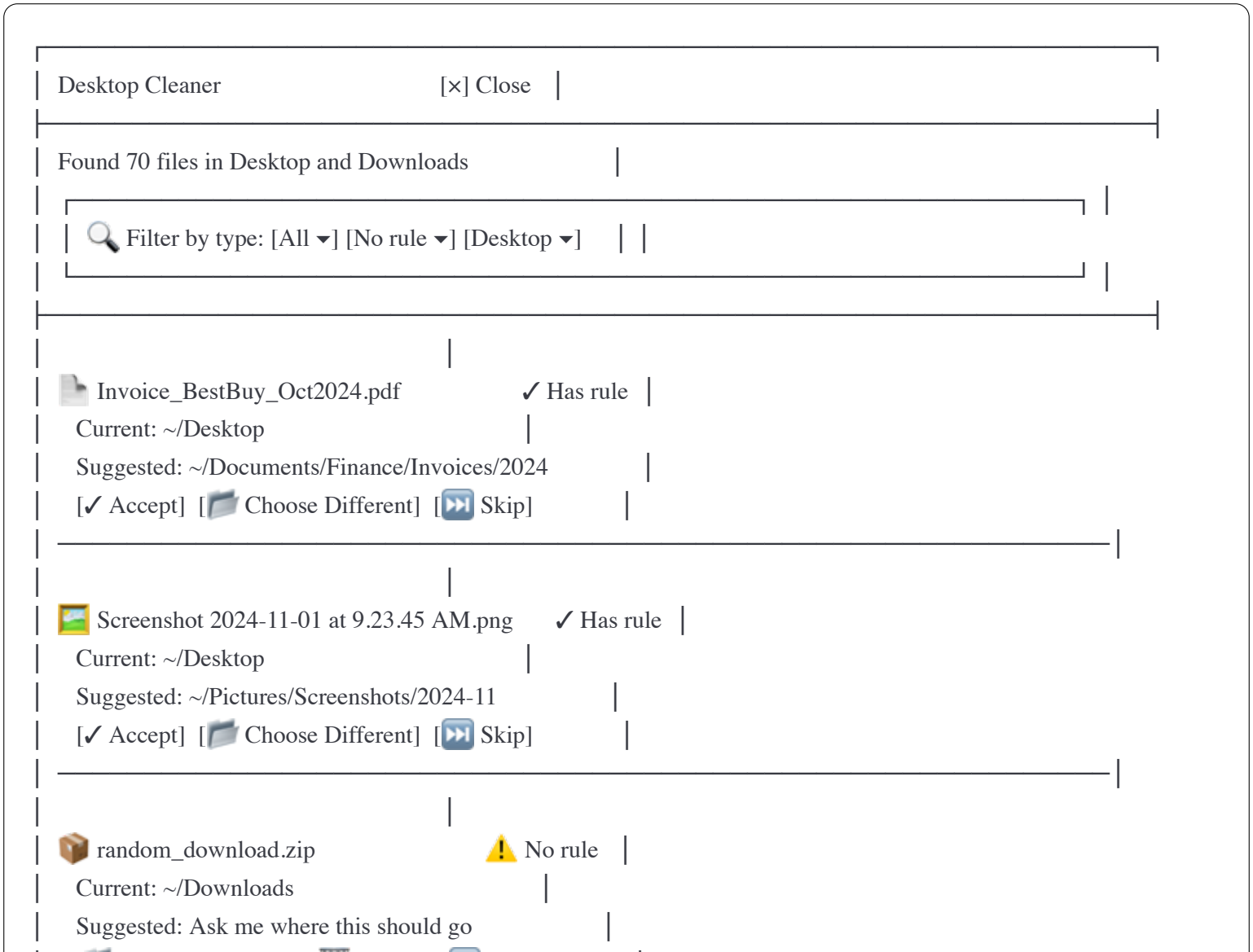
- **Normal:** Gray icon in menu bar
- **Files detected:** Badge with count (like Mail.app)
- **After cleanup:** Brief checkmark animation, return to normal

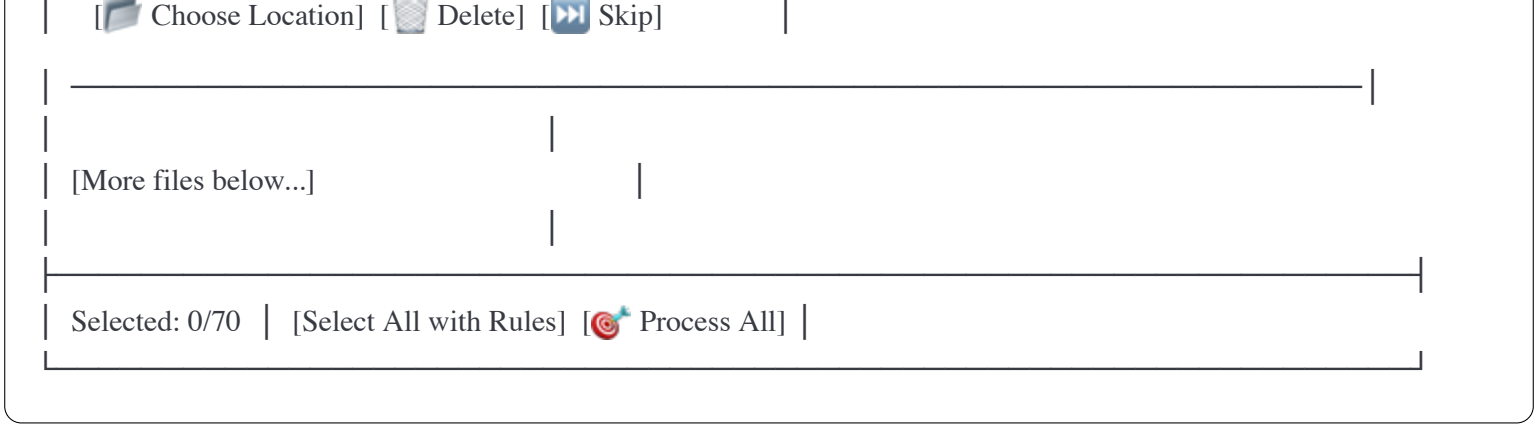
---

Screen 2: Review Interface (Main Window)

Purpose: Where 90% of interaction happens. Must be fast and keyboard-friendly.

Layout: List View (Recommended)





Key Features:

- One file per row with all info visible
- Status indicators:
  - ✓ Has matching rule
  - ⚠️ No rule found
  - 🤔 Uncertain match
- Inline actions (no modals for simple decisions)
- Batch operations at bottom

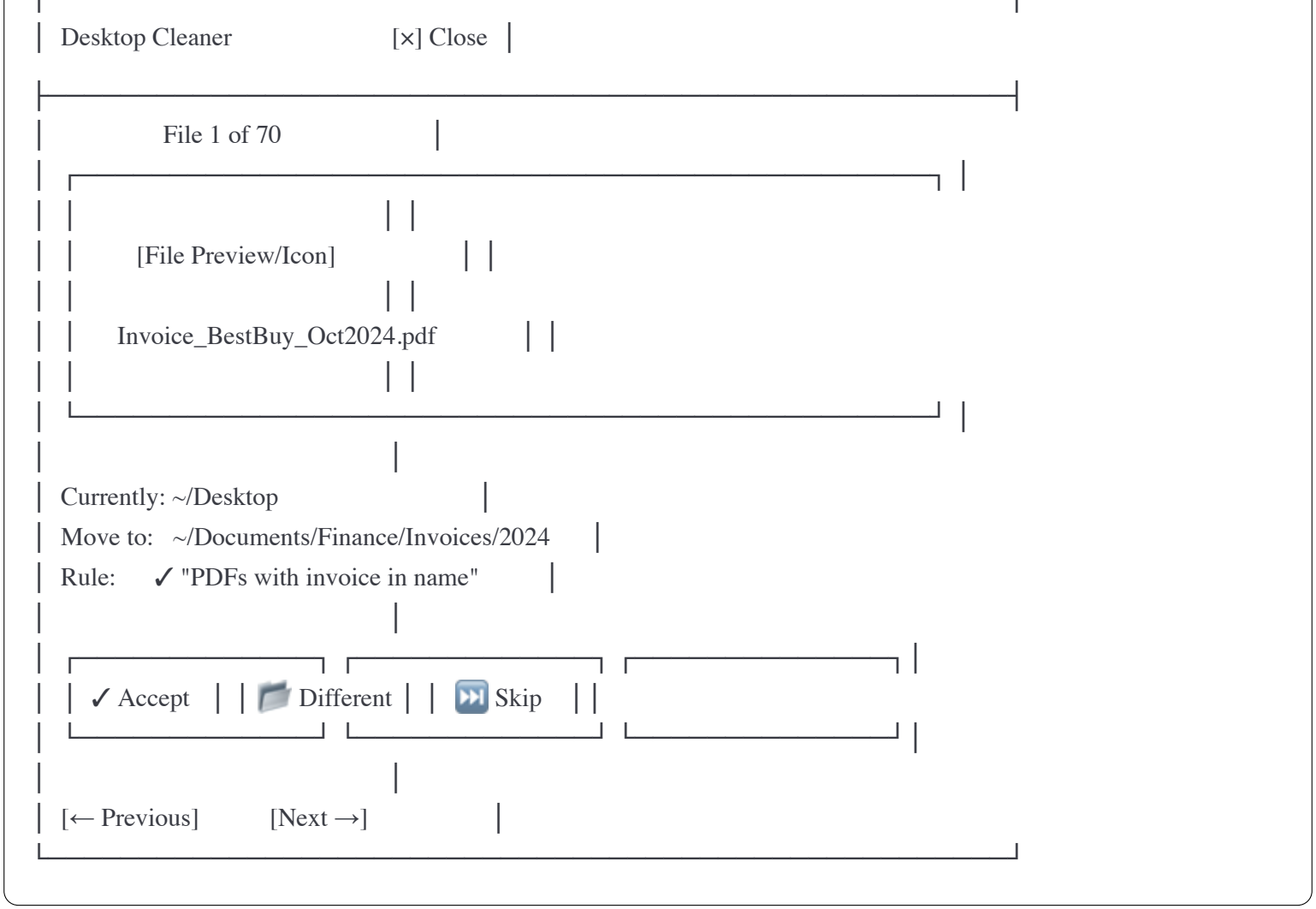
Keyboard Shortcuts:

- ⌘A = Accept current suggestion
- ⌘D = Choose different location
- ⌘Delete = Skip this file
- ↓ / ↑ = Navigate between files
- Space = Preview file (Quick Look)

Visual Hierarchy:

- Filename: Bold, 16pt
- Paths: Gray, 12pt, monospace font
- Buttons: Subtle until hover
- Rules matched: Small green checkmark
- No rules: Small orange warning icon

Alternative Layout: Card View



Card View Pros:

- More focused (one file at a time)
- Bigger preview area
- Less overwhelming
- Natural for swipe gestures

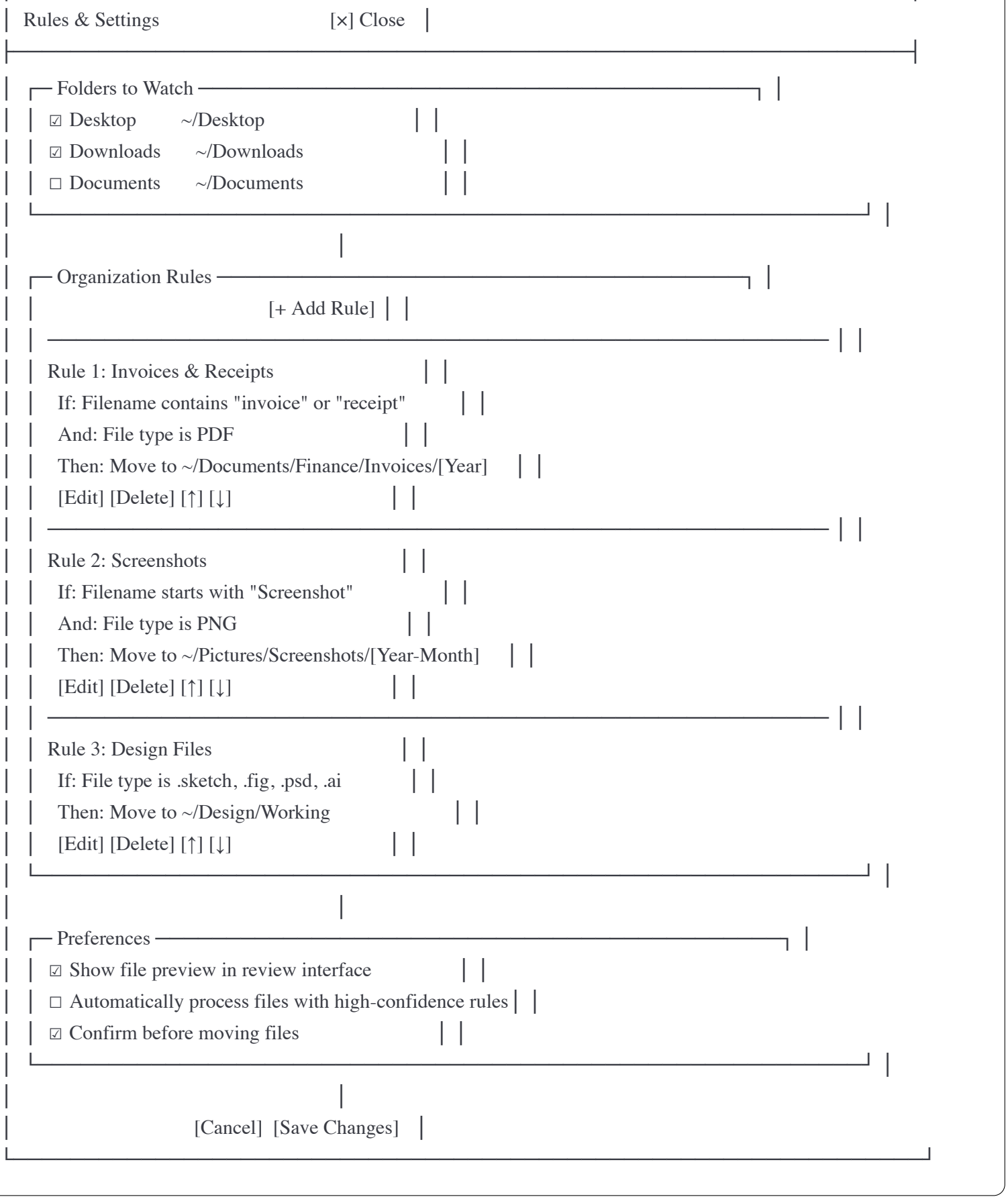
Card View Cons:

- Slower for bulk operations
- Can't see overview of all files

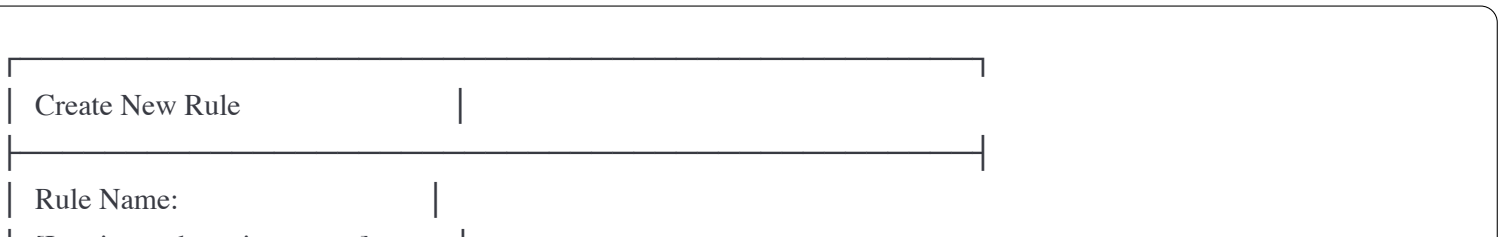
Recommendation: Start with List View for efficiency. Card view is more visual but less practical for processing many files quickly.

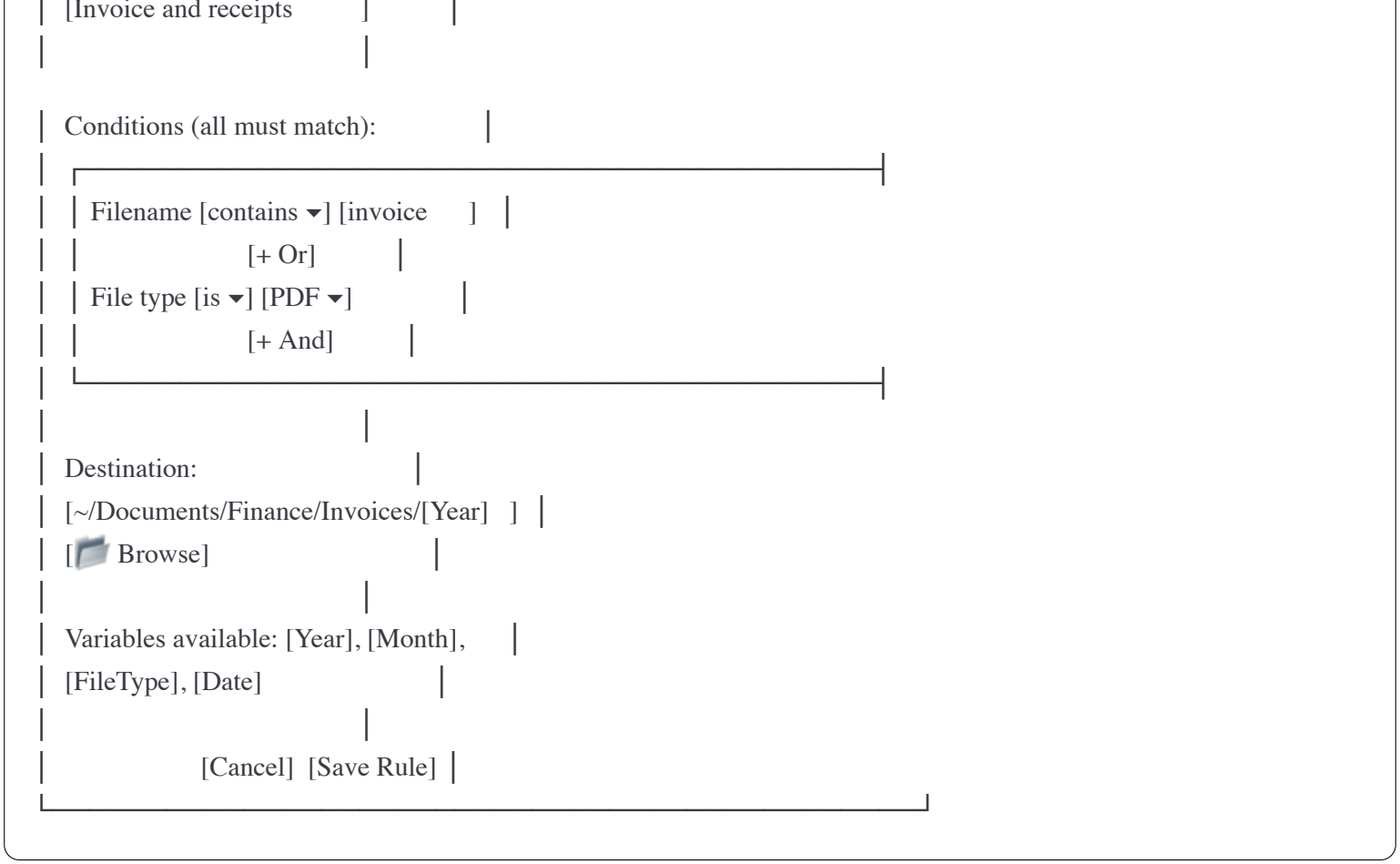
Screen 3: Rules & Settings

Purpose: Define organizational logic



Key Interactions - Adding a Rule:



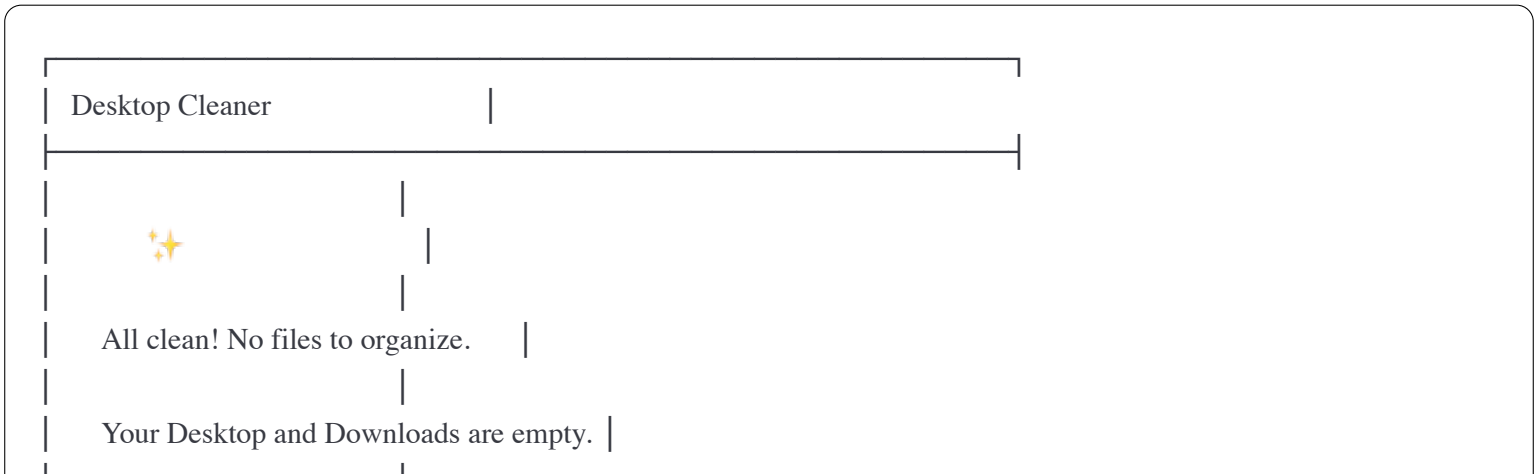


Rule Components:

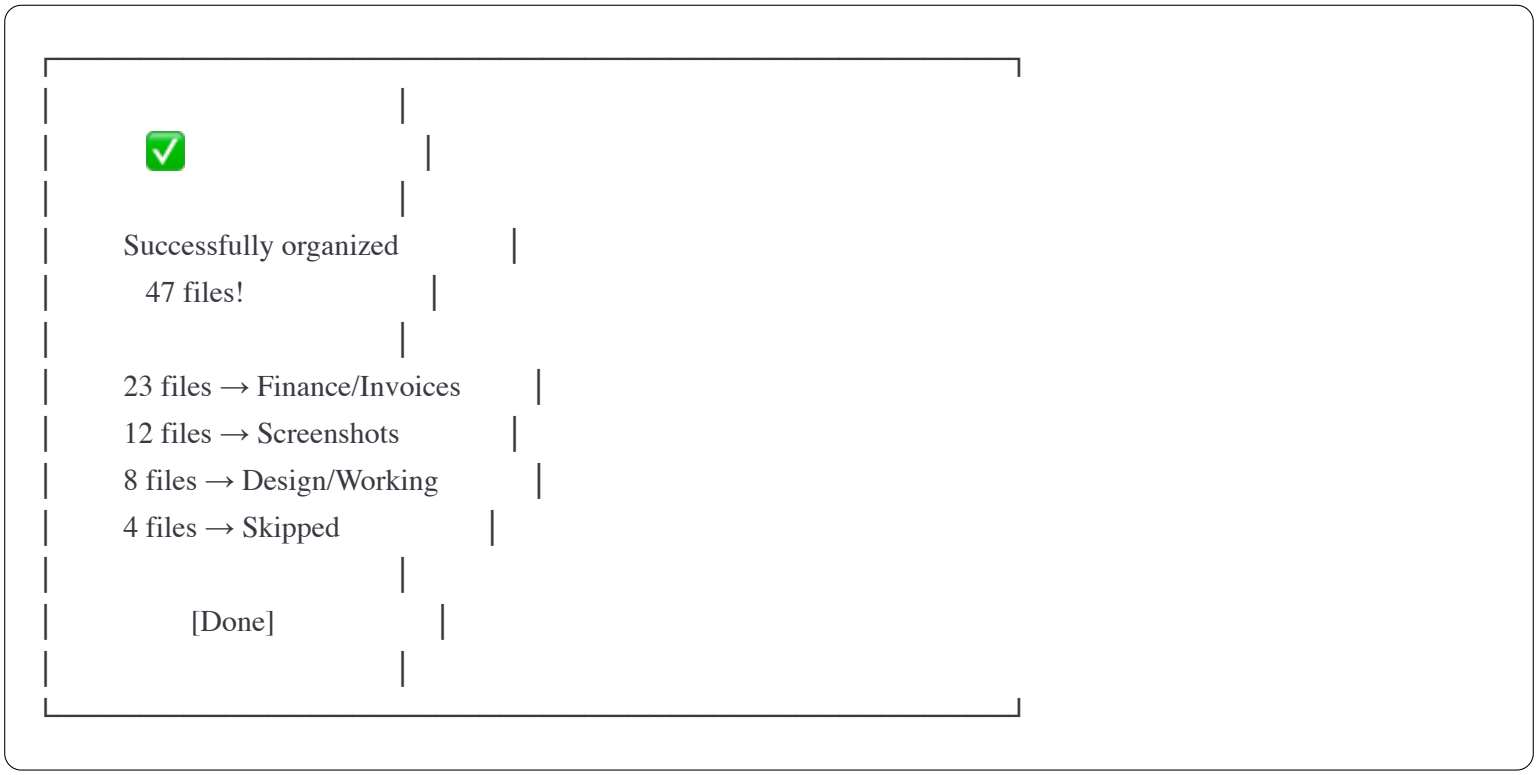
- **Conditions:** Multiple conditions with AND/OR logic
- **Operators:** contains, starts with, ends with, is, is not
- **File attributes:** Filename, file type, size, date modified, date created
- **Destination variables:** [Year], [Month], [Day], [FileType], [Date]
- **Priority:** Rules can be reordered (first match wins)

Supporting UI Elements

Empty State (No Files Found)



Success State (After Processing)



Interaction Flows

Happy Path: First-Time User

1. Launch app from menu bar
2. Grant permissions (folder access)
3. Open Settings → Create first rule
4. Click "Scan & Review"
5. See list of files with suggestions
6. Review and accept most, skip uncertain ones
7. Click "Process All"
8. Get success confirmation
9. Desktop is clean! 🎉

Power User Flow

1. Menu bar badge shows "47"
2. Click icon
3. Hit "Scan & Review"
4. Use keyboard shortcuts to blast through:
   - ↓ Space (preview) ⌘A (accept) = ~2 seconds per file
5. Complete in under 2 minutes

First Rule Creation Flow

1. Open Settings
2. Click "+ Add Rule"
3. Name the rule ("Invoices")
4. Add condition: Filename contains "invoice"
5. Add condition: File type is PDF
6. Set destination: ~/Documents/Finance/Invoices/[Year]
7. Save rule
8. Test with "Scan & Review"

Development Approach

Technology Stack (Recommended)

Primary: Swift + SwiftUI (Native macOS)

Why Swift/SwiftUI:

- Best performance for file system operations
- Full macOS integration (menu bar, notifications, Quick Look)
- Can add Core ML for AI features later
- Native look and feel
- Long-term foundation for quality tool

Learning Curve:

- 2-3 weeks to get comfortable with basics
- Different from React Native, but JavaScript knowledge transfers

- Different from React Native, but JavaScript knowledge transfers
- Plenty of free resources available

Alternative Options

Option 2: Electron

- Use familiar web technologies (JavaScript/React)
- Faster initial development
- Cross-platform potential
- **Cons:** Heavier memory footprint, less native feel

Option 3: Tauri

- Rust + web frontend
- Lighter than Electron, more native than web
- Relatively new but growing
- **Cons:** Smaller community, newer ecosystem

Recommendation: Swift/SwiftUI for building a quality Mac-native tool

---

30-Day Development Sprint (MVP)

Week 1: Learn & Setup

- ▢ Complete SwiftUI tutorial (Hacking with Swift recommended)
- ▢ Build "Hello World" menu bar app
- ▢ Learn FileManager basics (list files, move files)
- ▢ Set up Xcode project structure

Week 2: Core Engine

- ▢ Build file scanner (read Desktop/Downloads)
- ▢ Create simple rule engine (if filename.contains() → destination)
- ▢ Test file moving programmatically
- ▢ Handle permissions (Full Disk Access)

Week 3: UI Development

- ▢ Build review interface (list of files + suggestions)
- ▢ Add approve/reject actions
- ▢ Create settings screen to define rules
- ▢ Implement keyboard shortcuts

Week 4: Polish & Self-Test

- [ ] Menu bar integration

- [ ] Add empty states and success states
- [ ] Bug fixes and edge cases
- [ ] **Use it yourself for a week** - find rough edges
- [ ] Iteration based on real usage

Technical Considerations

macOS Permissions Required

- **Full Disk Access:** To scan and move files
- **File system monitoring:** FSEvents API (for Phase 2)
- **Notifications:** For nudges and reminders

Architecture Components

Menu Bar App Structure:

```swift
```

```swift
import SwiftUI

@main
struct DesktopCleanerApp: App {
    @NSApplicationDelegateAdaptor(AppDelegate.self) var appDelegate

    var body: some Scene {
        Settings {
            ContentView()
        }
    }
}

class AppDelegate: NSObject, NSApplicationDelegate {
    var statusItem: NSStatusItem?

    func applicationDidFinishLaunching(_ notification: Notification) {
        statusItem = NSStatusBar.system.statusItem(
            withLength: NSStatusItem.variableLength
        )
        if let button = statusItem?.button {
            button.image = NSImage(
                systemSymbolName: "folder.badge.gear",
                accessibilityDescription: "Desktop Cleaner"
            )
            button.action = #selector(menuBarItemClicked)
        }
    }

    @objc func menuBarItemClicked() {
        // Open main window
    }
}
```

File Scanner Basics:

```swift
func scanFolder(at path: String) -> [URL] {
```

```swift
    let fileManager = FileManager.default
    let folderURL = URL(fileURLWithPath: path)

    do {
        let files = try fileManager.contentsOfDirectory(
            at: folderURL,
            includingPropertiesForKeys: [.contentModificationDateKey],
            options: .skipsHiddenFiles
        )
        return files
    } catch {
        print("Error scanning: \(error)")
        return []
    }
}

// Usage
let desktopPath = NSHomeDirectory() + "/Desktop"
let desktopFiles = scanFolder(at: desktopPath)
```

Rule Engine Structure:

```swift
struct OrganizationRule {
    let id: UUID
    let name: String
    let conditions: [Condition]
    let destination: String
    let priority: Int
}

struct Condition {
    let attribute: FileAttribute  // .filename, .fileType, .size
    let operator: Operator        // .contains, .equals, .startsWith
    let value: String
}
```

```
enum FileAttribute {
    case filename
    case fileType
    case dateCreated
    case dateModified
    case size
}

enum Operator {
    case contains
    case equals
    case startsWith
    case endsWith
    case greaterThan
    case lessThan
}
```

Key Frameworks to Learn

- **SwiftUI:** UI framework
- **FileManager:** File operations
- **FSEvents:** File system monitoring (Phase 2)
- **Quick Look:** File previews
- **Core ML:** Machine learning (Phase 2)
- **UserDefaults / Core Data:** Saving rules and preferences

AI/ML Implementation (Phase 2)

Core ML Approach:

- Use **Create ML** (Apple's tool) to train text classifier on filenames
- Train on your actual organization patterns after using v0.1 for a month
- Runs entirely on-device (privacy win - no data leaves machine)
- Can classify with confidence scores

Training Process:

1. Export your organization history from MVP usage
2. Create training data: filename → destination category

3. Use Create ML to train classifier
4. Integrate trained model into app
5. Use for suggestions with confidence scoring

But don't think about this yet - build rule-based MVP first.

Design Language

For MVP: Clean & Mac-Native

Design Principles:

- Use **SF Symbols** (Apple's icon system) - free and consistent
- Use **system fonts** (San Francisco)
- Use macOS standard controls
- Match system **light/dark mode** automatically
- Minimal color usage initially
- Focus on clarity and speed

Personality: Utility-focused

- Like Hazel: Doesn't get in your way
- Quiet, efficient, reliable
- No unnecessary animation or flourishes in MVP

Future Visual Identity (Phase 2+)

After MVP proves valuable, consider:

- Custom iconography (leverage your 3D skills in Blender)
- Refined color palette
- Custom animations
- Branded empty states
- Delight moments (subtle celebrations after cleanup)

Personality Options to Explore:

- **Utility-focused:** Gray, minimal, invisible helper

- **Friendly assistant:** Touch of color, encouraging, approachable
- **Premium tool:** Refined typography, spacious, beautiful

Business Model

Freemium Structure

Free Tier:

- Manual rule creation (unlimited rules)
- Desktop + Downloads scanning
- Basic file type sorting
- Review interface with manual approval
- Up to 100 files per scan

Paid Tier ($4.99/month or $49/year):

- AI-powered suggestions and learning
- Unlimited files per scan
- Duplicate detection
- Archive management
- Background monitoring and nudges
- Priority support
- Early access to new features

Why This Model:

- Free tier proves value and gets people hooked
- AI features justify paid tier (real computational value)
- Indie developer sustainable pricing
- Alternative: One-time purchase ($29-49) for simpler model

Market Position

Competitors:

- **Hazel:** $42 one-time, powerful but complex, no AI
- **CleanMyMac:** $40/year, cleanup but not organization

- **Default Folders X:** $35 one-time, helps during save but not cleanup

Your Advantage:

- AI learns YOUR specific patterns
- Modern, delightful UX (not power-user-only)
- Balance of automation and control
- Privacy-focused (on-device processing)
- Built for 2025+ macOS users

Resources

Learning Swift/SwiftUI

- Hacking with Swift - 100 Days of SwiftUI - Free, comprehensive
- Apple's SwiftUI Tutorials - Official documentation
- SwiftUI by Example - Quick reference guide

Menu Bar Apps

- Creating a macOS Menu Bar App Tutorial
- Menu Bar Extra in SwiftUI

File Management

- FileManager Documentation
- Working with Files in Swift

Core ML (For Phase 2)

- Create ML Documentation
- Core ML Overview

Next Steps

Immediate Actions (Choose One Path)

Path 1: UI Prototyping First

1. **Sketch on paper** (30 mins) - Quick iteration
2. **Figma mockup** (2-3 hours) - Your macOS UI kit

2. **Figma mockups** (2-3 hours) - Use macOS UI kits
3. **Get feedback** from friends/colleagues
4. **Refine based on feedback**

Path 2: Learn & Build Simultaneously

1. **Start Swift/SwiftUI tutorial** (Week 1 goal)
2. **Build static UI** with hardcoded data
3. **See what's easy vs. hard** to build
4. **Adjust design** based on technical constraints

Path 3: Technical Exploration

1. **Prototype file scanner** in Swift (get basic working)
2. **Test file moving** (ensure permissions work)
3. **Validate core assumptions** about file operations
4. **Then design UI** around what's technically feasible

Recommended: Path 2 (Learn & Build)

- Spend 1 hour sketching UI on paper
- Start SwiftUI tutorial
- Build UI with static/fake data
- Learn what's easy to build vs. hard
- Let technical reality inform design decisions

90-Day Roadmap

Month 1: MVP Development

- Learn Swift/SwiftUI fundamentals
- Build core file scanner and rule engine
- Create basic UI (list view + settings)
- Get it minimally working

Month 2: Self-Testing & Refinement

- Use the app yourself daily
- Fix bugs and edge cases

- Add keyboard shortcuts and polish
- Refine rules based on real usage

Month 3: Feature Complete MVP

- Add empty states and success feedback
- Implement proper error handling
- Write basic documentation
- Prepare for potential beta testers

Design Decisions Log

Decisions Made

List View vs. Card View:

- **Decision:** Start with List View
- **Reasoning:** Need to process many files quickly; overview is valuable
- **Reconsider when:** Phase 2 if user testing shows preference for focused view

Native Swift vs. Electron:

- **Decision:** Swift/SwiftUI
- **Reasoning:** Building quality Mac-native tool; long-term foundation; can add Core ML
- **Trade-off:** Longer learning curve, but worth it for final quality

Free vs. Paid:

- **Decision:** Freemium model
- **Reasoning:** Free tier proves value; AI features justify paid tier
- **Alternative considered:** One-time purchase ($39)

MVP Scope:

- **Decision:** Manual rules only, Desktop + Downloads, no AI
- **Reasoning:** Prove core concept works before adding complexity
- **Add later:** AI learning, background monitoring, archive features

Open Questions

Name & Branding:

- Not decided yet
- Will emerge naturally after using the tool
- Consider after MVP is functional

Visual Identity:

- Start Mac-native/minimal
- Add personality after core experience is solid
- Leverage 3D skills (Blender) for custom iconography later

Distribution:

- Self-hosted initially (personal use)
- Consider Mac App Store vs. direct sales later
- Gumroad/Paddle for indie-friendly payment processing

Success Criteria

MVP Success = Personal Usage

- Successfully organizing own Desktop/Downloads weekly
- Feeling less overwhelmed by file clutter
- Rules working 80%+ of the time
- Speed improvement over manual organization
- **Qualitative win:** "I actually want to use this"

Phase 2 Success = Others Want It

- Beta testers actively using it
- Positive feedback on core concept
- People willing to pay for AI features
- Testimonials: "This solved my problem"

Long-term Success = Sustainable Product

- 1,000+ active users
- 30%+ conversion to paid tier

- Sustainable indie developer income
- Feature requests and engagement
- Becoming "the" tool for Mac desktop organization

Appendices

File Types to Handle

Documents:

- PDF, DOC, DOCX, TXT, RTF, PAGES
- XLS, XLSX, CSV, NUMBERS
- PPT, PPTX, KEYNOTE

Images:

- JPG, JPEG, PNG, GIF, HEIC
- SVG, WEBP, TIFF
- PSD, AI, SKETCH, FIG (design files)

Code:

- JS, JSX, TS, TSX, PY, SWIFT
- HTML, CSS, JSON, YAML
- Project folders (node_modules, etc.)

Archives:

- ZIP, RAR, 7Z, TAR, GZ
- DMG, PKG, APP

Media:

- MP4, MOV, AVI, MKV (video)
- MP3, WAV, M4A (audio)

Potential Rule Examples

Invoice & Receipt Rule:

- If: Filename contains "invoice" OR "receipt"
- And: File type is PDF

- Then: ~/Documents/Finance/Invoices/[Year]

Screenshot Rule:

- If: Filename starts with "Screenshot"
- And: File type is PNG
- Then: ~/Pictures/Screenshots/[Year-Month]

Design Work Rule:

- If: File type is .sketch, .fig, .psd, .ai
- Then: ~/Design/Working/[Date]

Code Project Rule:

- If: Filename ends with .zip
- And: Filename contains "github" OR "project"
- Then: ~/Code/Archives/[Year]

Client Work Rule:

- If: Filename contains [ClientName]
- And: File type is PDF or DOCX
- Then: ~/Work/Clients/[ClientName]/[Year]

Tech Stack Summary

Languages & Frameworks:

- Swift 5.9+
- SwiftUI for UI
- Combine for reactive programming

Apple Frameworks:

- Foundation (FileManager, URL, etc.)
- AppKit (Menu bar, windows)

- Quick Look (File previews)
- Core ML (Future: AI features)
- FSEvents (Future: File monitoring)

Tools:

- Xcode 15+
- Create ML (Future: Model training)
- Git for version control

Testing:

- XCTest for unit tests
- Manual QA (yourself as primary user)

Project Timeline

Total Time to Usable MVP: 3-6 months (part-time)

Milestones:

- **Week 4:** Basic file scanner working
- **Week 8:** UI functional with static data
- **Week 12:** Can actually move files with rules
- **Week 16:** Using it yourself regularly
- **Week 20:** Polished enough for close friends to test
- **Week 24:** Decide if this becomes a product or stays personal

---

Reflection & Notes

Why This Project Matters:

- Solves real personal pain point
- Combines technical learning with practical tool building
- Potential for sustainable indie product
- Fits skill development goals (Swift, macOS, ML)
- Could help others with same organization struggles

Risk Factors:

- Learning new language/framework simultaneously
- Scope creep (must resist adding features too early)