Capcat TUI/CLI - Ethical Web Scraper

Short Case Study

Role: Product Designer and MVP Developer Timeline: 2.5 months, end of 2025 Team: Solo project

---

Summary

Capcat is a free, open-source command-line tool for ethical content archiving. Built solo over 2.5 months, it ships two interfaces - a visual TUI and a flag-based CLI - on one shared backend. The tool is live at capcat.org.

The process was grounded in UX research, Jobs-to-be-Done, heuristic analysis against Nielsen's 10 principles and Laws of UX, and a disciplined spec-driven LLM development cycle. The project also served as a deliberate exercise in context engineering - structuring what an LLM receives before generation to directly control the quality of its output.

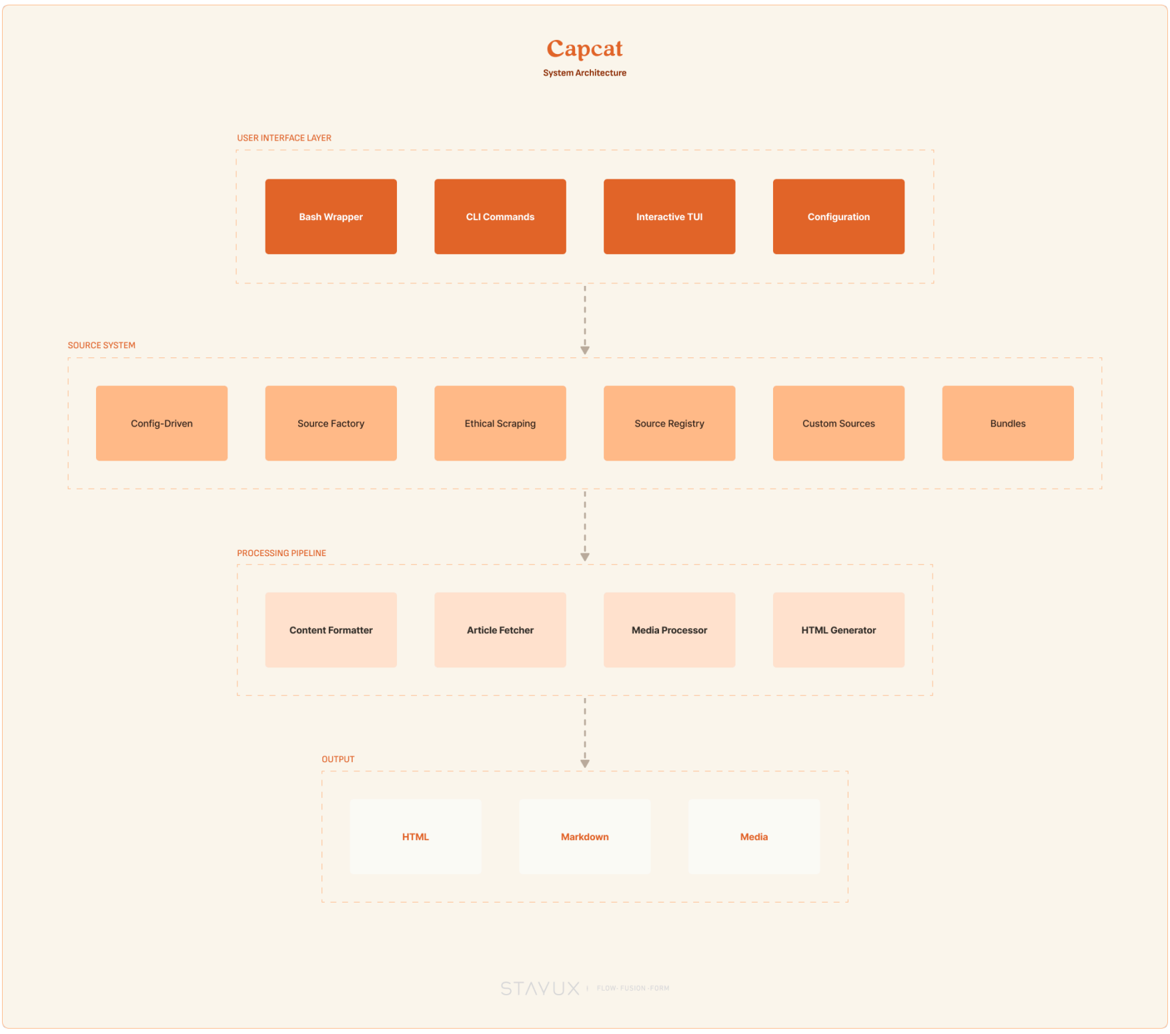
Capcat
System Architecture
USER INTERFACE LAYER
Bash Wrapper
CLI Commands
Interactive TUI
Configuration
SOURCE SYSTEM
Config-Driven
Source Factory
Ethical Scraping
Source Registry
Custom Sources
Bundles
PROCESSING PIPELINE
Content Formatter
Article Fetcher
Media Processor
HTML Generator
OUTPUT
HTML
Markdown
Media
STAVUX

Figure 1: System Architecture

Challenge and Context

Reading across multiple sources weekly had produced scattered archives with no connection between them. Content disappeared. Bookmarks gave no context. Search inside personal archives was unreliable.

No existing tool solved all four jobs simultaneously: archive content before it disappears, catalogue it for search, retrieve it on demand, and share it without depending on an external service. Every tool solved one part. None solved the combination.

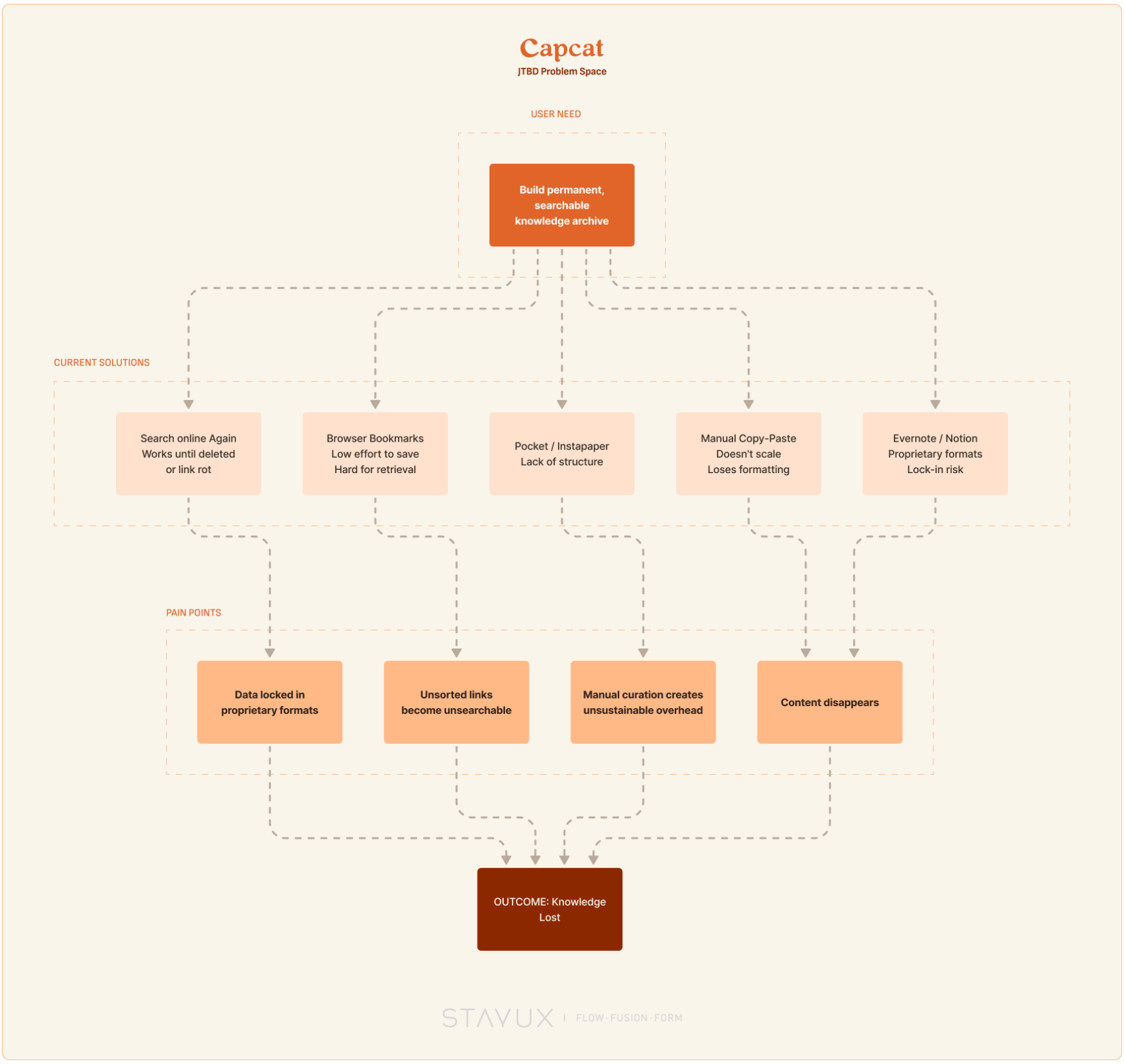
Capcat
JTBD Problem Space
USER NEED
Build permanent, searchable knowledge archive
CURRENT SOLUTIONS
Search online Again
Works until deleted or link rot
Browser Bookmarks
Low effort to save
Hard for retrieval
Pocket / Instapaper
Lack of structure
Manual Copy-Paste
Doesn't scale
Loses formatting
Evernote / Notion
Proprietary formats
Lock-in risk
PAIN POINTS
Data locked in proprietary formats
Unsorted links become unsearchable
Manual curation creates unsustainable overhead
Content disappears
OUTCOME: Knowledge Lost
STAVUX

Figure 2: JTBD Problem Space

Discovery and User Insights

Research started with self-observation - mapping my own workflow before writing a single requirement. Heuristic analysis followed. Informal conversations with three people (a developer, a researcher, an ML engineer) confirmed the core job-to-be-done without priming them: two independently raised the problem of content disappearing from original sources.

Four user groups share the same gap - no reliable path from intent to content:

- **Design engineers** want automated content collection for information pipelines
- **Developers** want searchable local copies independent of external availability
- **ML engineers** want structured, parseable output for tools and agents
- **CLI-familiar users** want flags, pipes, and scripts with no GUI required

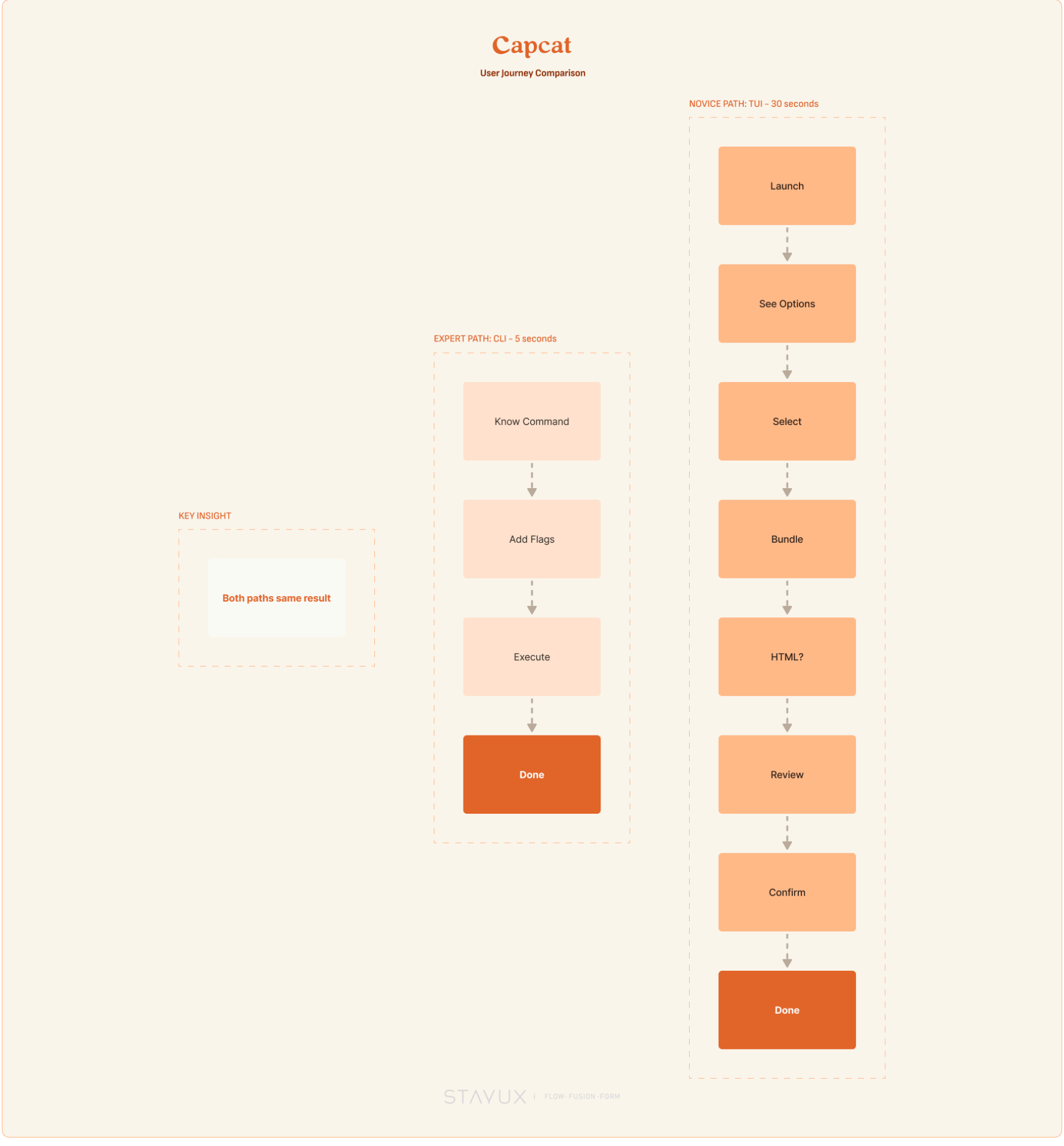
Capcat
User Journey Comparison
NOVICE PATH: TUI - 20 seconds
Launch
See Options
Select
Bundle
HTML?
Review
Confirm
Done
EXPERT PATH: CLI - 5 seconds
Know Command
Add Flags
Execute
Done
KEY INSIGHT
Both paths same result
STAVUX

Figure 3: User Journey Comparison

Information Architecture and Strategy

The first version - called ScanGrab - was a single scraping command with no error handling and no way to add sources without editing code. After a full JTBD and heuristic pass, it was scrapped. The name changed to Capcat: Cap(ture) Cat(alogue), encoding the two core jobs.

Three non-negotiable constraints were locked in the PRD:

1. **The archive belongs to the user.** All output is self-contained files with no server dependency.
2. **The tool meets users where they are.** A guided TUI and a flag-based CLI on one backend - neither is a reduced version of the other.
3. **Ethical scraping is built in, not opted in.** The EthicalScrapingManager sits at the architecture level. Every source passes through it before touching the network.

The folder structure was designed as information architecture - legible to a file manager, Obsidian, a local search tool, or a script, without Capcat acting as intermediary. The archive is self-documenting: structure alone tells you what is in it and when it arrived.

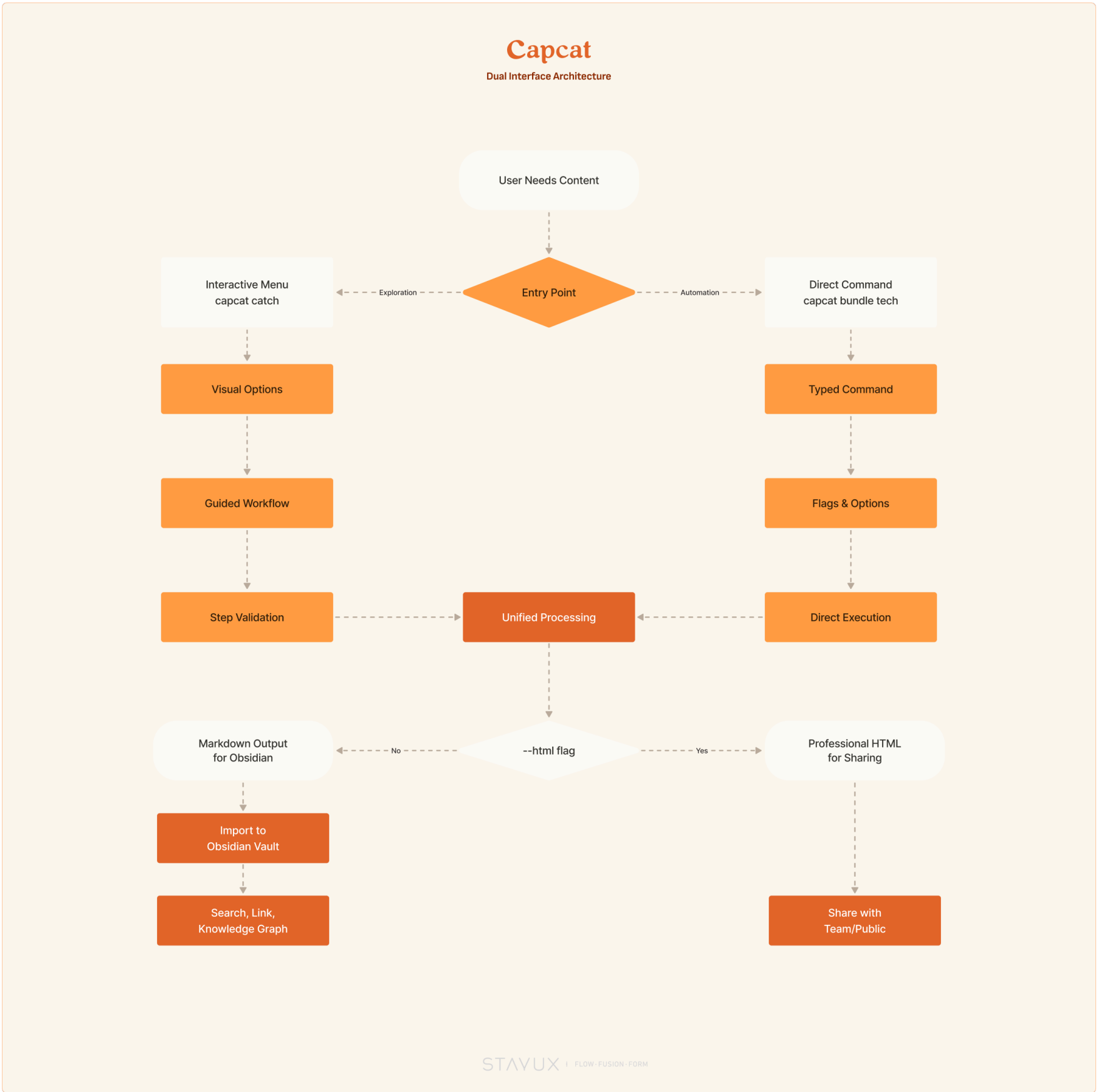
Capcat
Dual Interface Architecture
User Needs Content
Entry Point
Exploration
Automation
Interactive Menu
capcat catch
Direct Command
capcat bundle tech
Visual Options
Typed Command
Guided Workflow
Flags & Options
Step Validation
Unified Processing
Direct Execution
Markdown Output
for Obsidian
--html flag
No
Yes
Professional HTML
for Sharing
Import to
Obsidian Vault
Search, Link,
Knowledge Graph
Share with
Team/Public
STAVUX

Figure 4: Dual Interface Architecture

Iterative Design and Testing

The CLI vocabulary was designed before any code was written. Every command follows a verb-first structure: `fetch`, `bundle`, `single`, `list`, `catch`. The reference standard was clig.dev. Users familiar with git or docker have a rough working model of Capcat before reading any documentation.

The TUI was built as a separate surface on the same processing core - not a wrapper over the CLI. The novice path walks eight steps; the expert path takes three. Both produce identical output.

Heuristics shaped every interaction decision:

- **H1 - Visibility of System Status:** Every fetch reports progress in real time.
- **H6 - Recognition Over Recall:** The full main menu appears on launch. Nothing requires memorization.
- **H3 - User Control and Freedom:** Ctrl+C returns to the menu at any point. Every sub-menu carries an explicit exit. No dead ends.
- **H7 - Flexibility and Efficiency:** The CLI is the accelerator layer the novice never needs but the power user depends on.

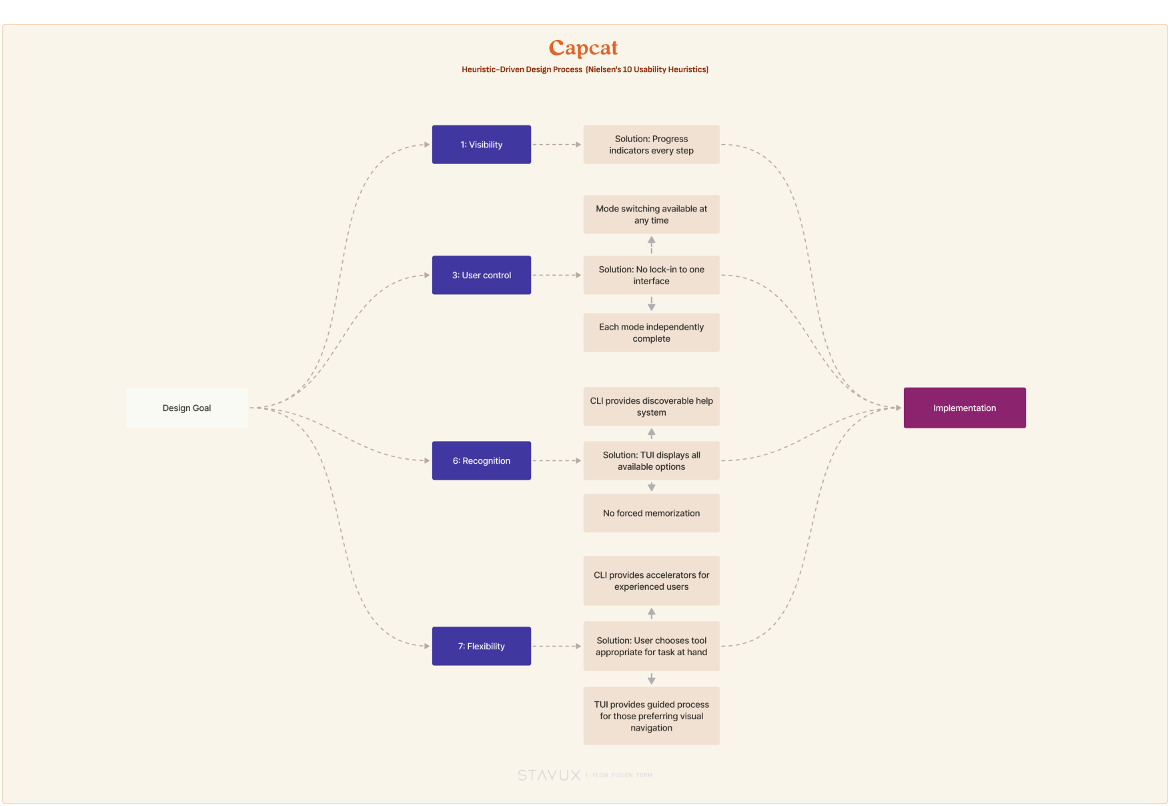
Capcat
Heuristic-Driven Design Process (Nielsen's 10 Usability Heuristics)
Design Goal
1: Visibility
Solution: Progress indicators every step
Mode switching available at any time
3: User control
Solution: No lock-in to one interface
Each mode independently complete
CLI provides discoverable help system
6: Recognition
Solution: TUI displays all available options
No forced memorization
CLI provides accelerators for experienced users
7: Flexibility
Solution: User chooses tool appropriate for task at hand
TUI provides guided process for those preferring visual navigation
Implementation
STAVUX

Figure 5: Heuristic-Driven Design Process

The Final Solution

Three primary workflows cover the full range of use:

Bundles - predefined source groups (tech, news, science, ai, sports). Select a bundle, choose HTML output, confirm, execute. One decision tree, one result.

Single article - paste any URL. Capcat detects the source, applies the right handler, falls back to generic scraping when needed. The user sees none of this. Output is always clean Markdown with downloaded images.

Source management - add any RSS feed, validate it, assign it a category, optionally add it to a bundle, optionally run a test fetch. Complex sites get an interactive wizard that generates a full YAML config with selectors, rate limiting, and skip patterns.

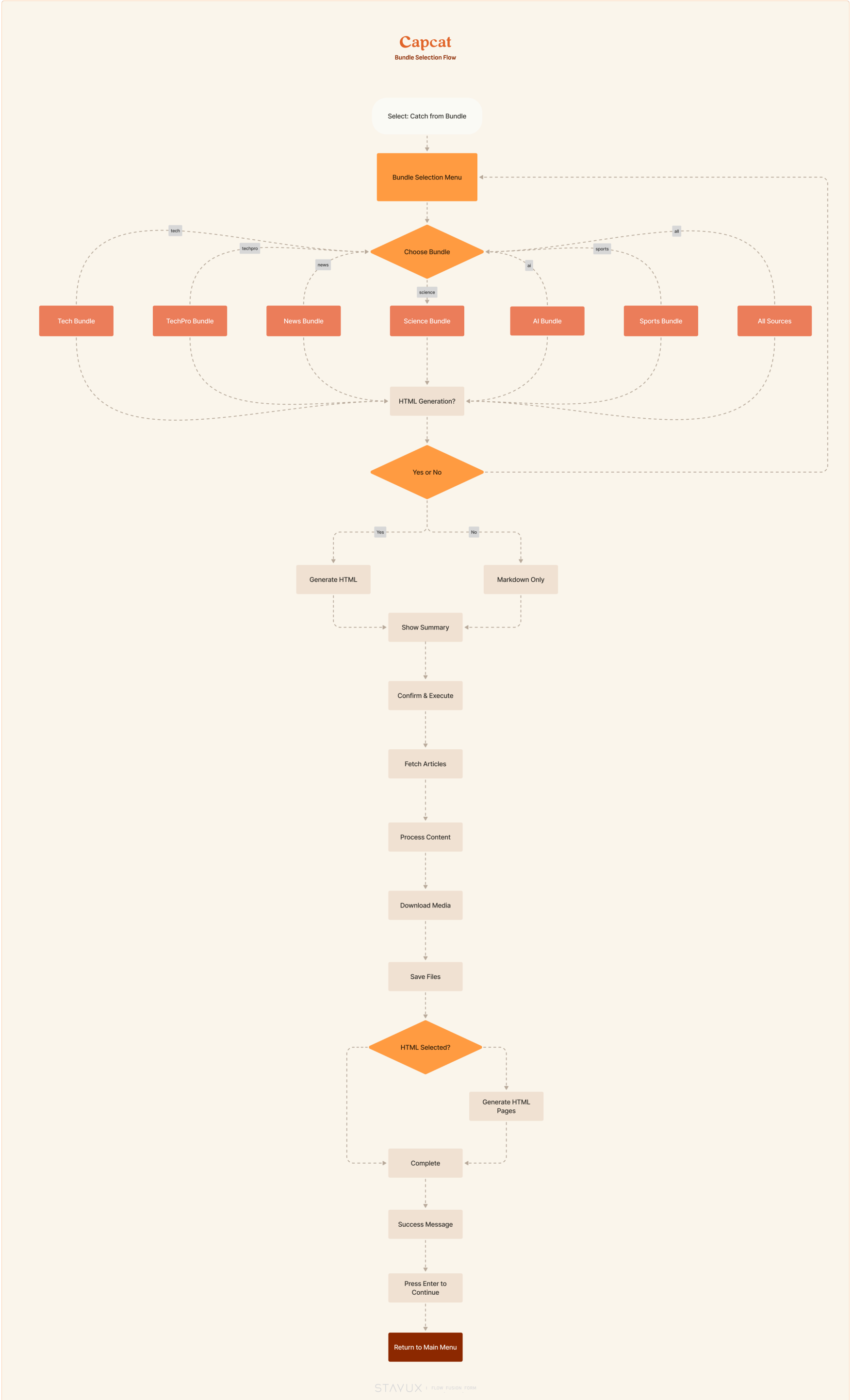
Capcat
Bundle Selection Flow
Select: Catch from Bundle
Bundle Selection Menu
Choose Bundle
tech
techpro
news
science
ai
sports
all
Tech Bundle
TechPro Bundle
News Bundle
Science Bundle
AI Bundle
Sports Bundle
All Sources
HTML Generation?
Yes or No
Yes
No
Generate HTML
Markdown Only
Show Summary
Confirm & Execute
Fetch Articles
Process Content
Download Media
Save Files
HTML Selected?
Generate HTML Pages
Complete
Success Message
Press Enter to Continue
Return to Main Menu

Figure 6: Bundle Selection Flow

HTML Output

The `--html` flag generates self-contained output: embedded CSS and JavaScript, no external dependencies, no CDN, no server required. A folder sent to someone else opens completely in a browser offline.

Design decisions were evaluated against one constraint: does this work when the folder is closed and reopened on a different machine with no internet?

Key design choices:

- **Editorial structure** - the index presents each day's fetch like a newspaper front page: sources as sections, articles as titled entries with metadata.
- **Reading surface** - constrained line length, serif headings at light weight, reading progress bar in the sticky header, navigation at top and bottom of every article.
- **Theme persistence** - dark/light toggle is a one-time user choice that propagates across the entire archive.
- **Comment hierarchy** - left border tones shift with depth, making thread nesting legible at a glance without counting indent levels.
- **Author anonymization** - built into the write step, not offered as a setting.
- **Code syntax themes** - two distinct palettes (One Dark for dark mode, GitHub convention for light) rather than one inverted palette.
- **PDF integration** - linked reference bar at the top of any article that has associated PDFs, stored locally in the article folder.

The design system is fully open: one `design-system.css` file. Edit it once, every article updates on the next fetch.

Validation and Impact

Interface decisions were validated against three cognitive laws:

Hick's Law - decision time increases with number of choices. The main menu holds six options. Sub-menus add two or three, never a wall. Complexity reveals itself progressively as the user goes deeper.

Miller's Law - working memory holds roughly four to seven items. The CLI has nine primary commands - slightly above threshold, but each is a single verb mapping to a single action, targeting users who routinely hold more in working memory.

Jakob's Law - users bring mental models from other tools. `capcat fetch hn,bbc` maps to `git fetch`. `capcat bundle tech` maps to `git bundle`. `capcat list sources` maps to `docker ps --list`. The command vocabulary was chosen to transfer existing knowledge rather than require new learning.

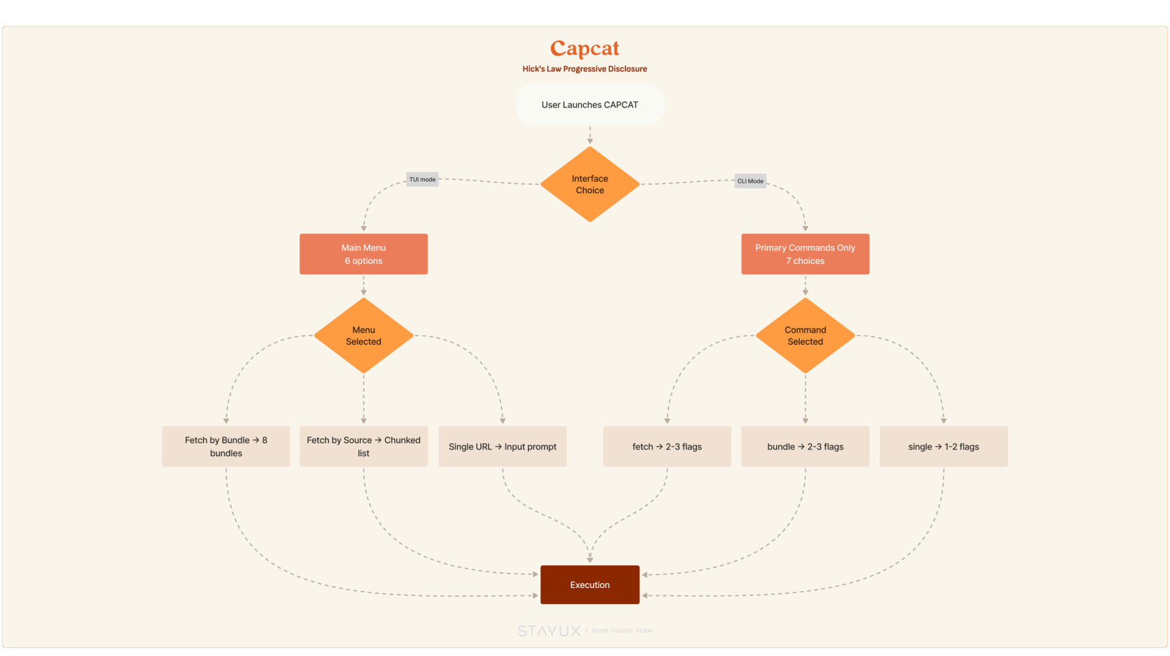
Capcat
Hick's Law Progressive Disclosure
User Launches CAPCAT
Interface Choice
TUI mode
CLI Mode
Main Menu
6 options
Primary Commands Only
7 choices
Menu Selected
Command Selected
Fetch by Bundle → 8 bundles
Fetch by Source → Chunked list
Single URL → Input prompt
fetch → 2-3 flags
bundle → 2-3 flags
single → 1-2 flags
Execution
STAVUX

Figure 7: Hick's Law Progressive Disclosure

Retrospective and Learnings

Ethical scraping is a constraint the architecture enforces, not a feature the user configures. The EthicalScrapingManager sits at the same level as Source Factory and Source Registry in the system architecture. Every source passes through it before touching the network. Robots.txt is cached, rate limiting is enforced per domain at one request per ten seconds, and the user agent identifies itself honestly.

The user never has to wonder whether Capcat is behaving lawfully. That uncertainty is removed by design.

Two learnings with broad application:

- A CLI and a TUI are two separate design problems that share a backend.
- Ethical constraints belong in the architecture, not in the documentation.

Product Website

capcat.org serves three audiences: a user evaluating the tool, a developer reading architecture docs, and a contributor finding the GitHub link and ethical scraping guidelines.

The design system uses CSS variables with a system font stack and no external dependencies. The color palette is a nine-step orange scale - the same orange used in terminal output coloring, so the brand and the interface share one identity. Eight custom SVG icons illustrate the feature section, one per capability.

Branding and Illustration

The name compresses four meanings into four letters: Cap(ture), Cat(alogue), the mascot cat, and `cat` the Unix command. It works on first encounter and holds more the longer you look at it.

The logotype is a handmade serif drawn by hand, built on a precise construction grid. The aperture of the capital C curves in a way that reads as a cat's tail - the mascot is present in the first letter without being illustrated. The lowercase descender on the p echoes it a second time.

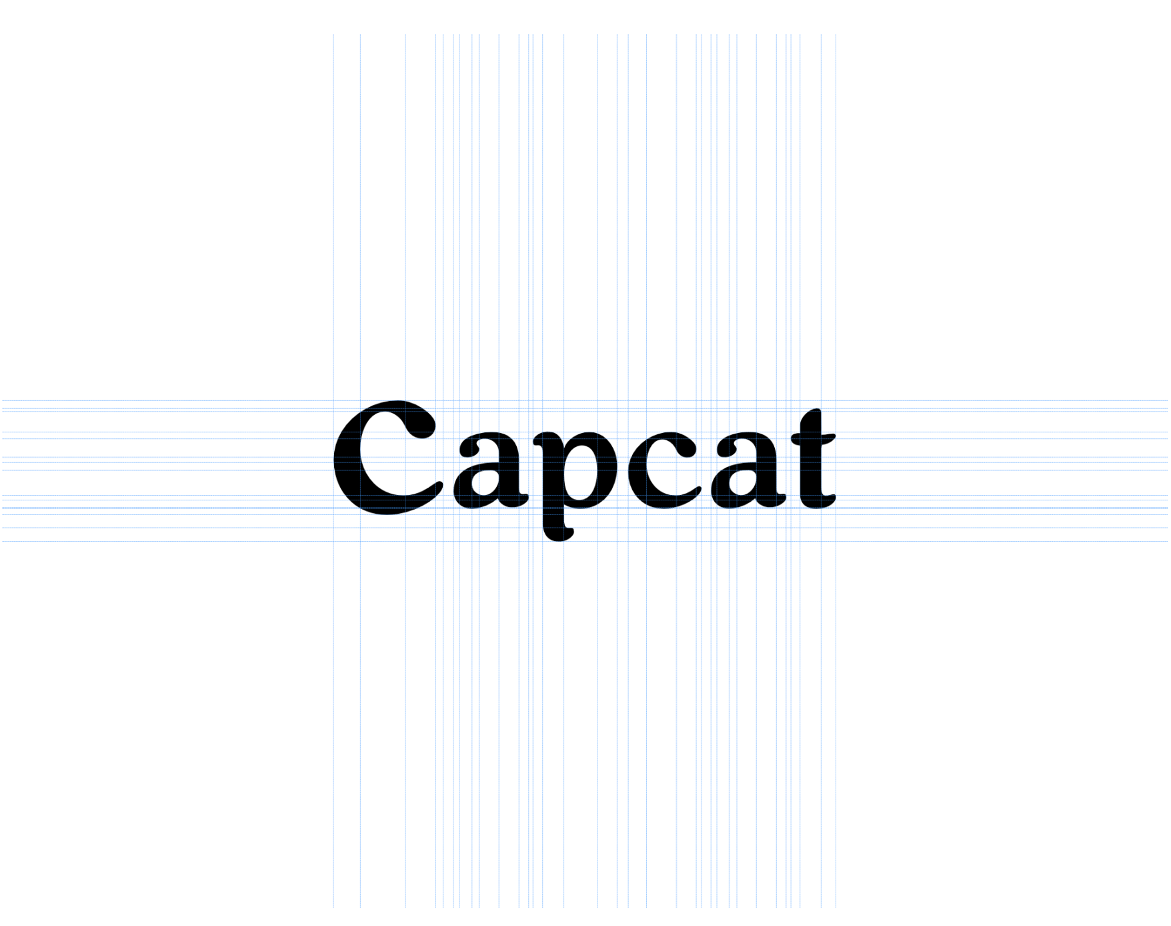

Figure 8: Logotype Construction Grid

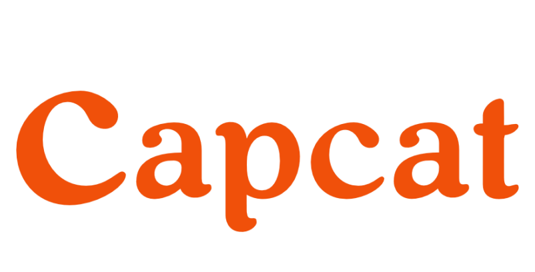

Figure 9: Logotype Brand Color

The mascot is a cat dressed as a baseball player, catching a loading ball from a progress bar while a crowd of computers cheers from the stands. Drawn by hand on paper, refined in Procreate, vectorized in Affinity Designer.

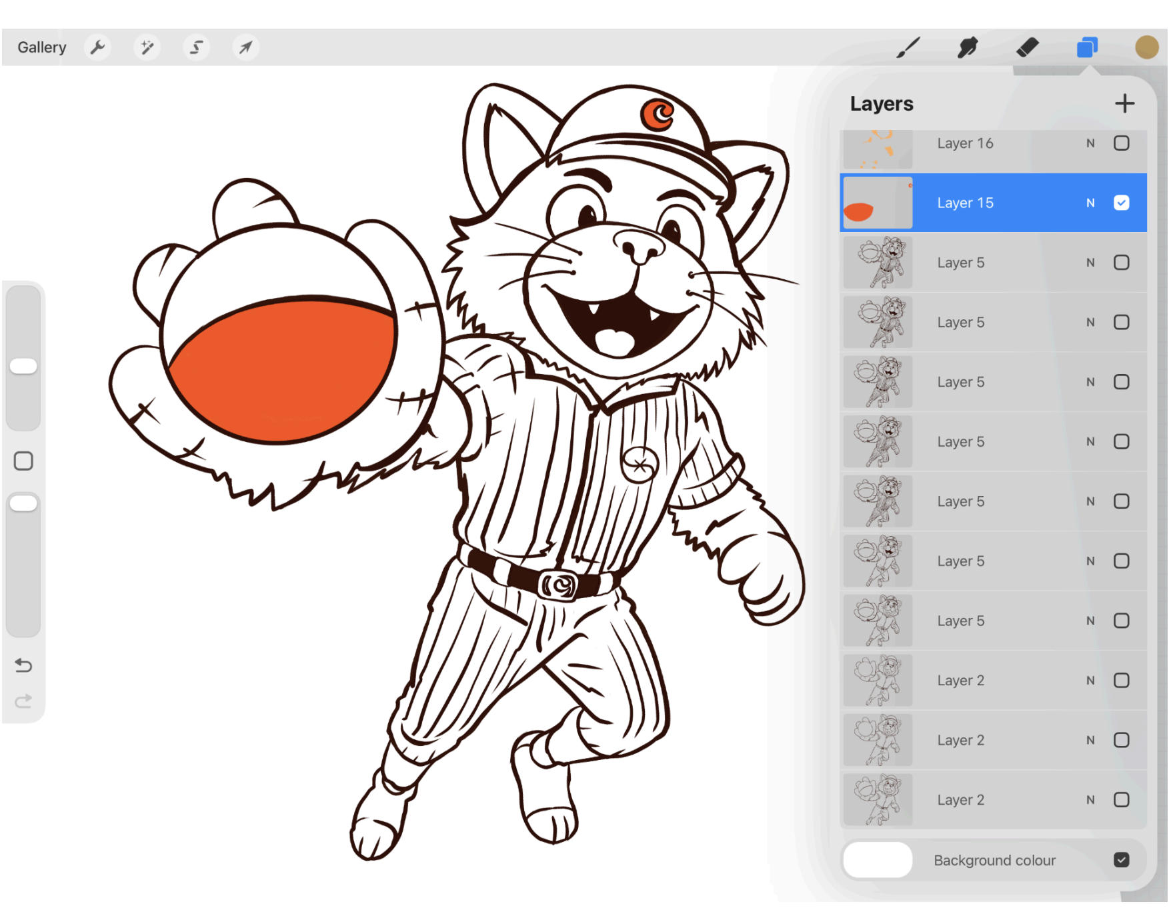
Gallery
Layers
Layer 16
Layer 15
Layer 5
Layer 2
Background colour

Figure 10: Capcat Illustration 1

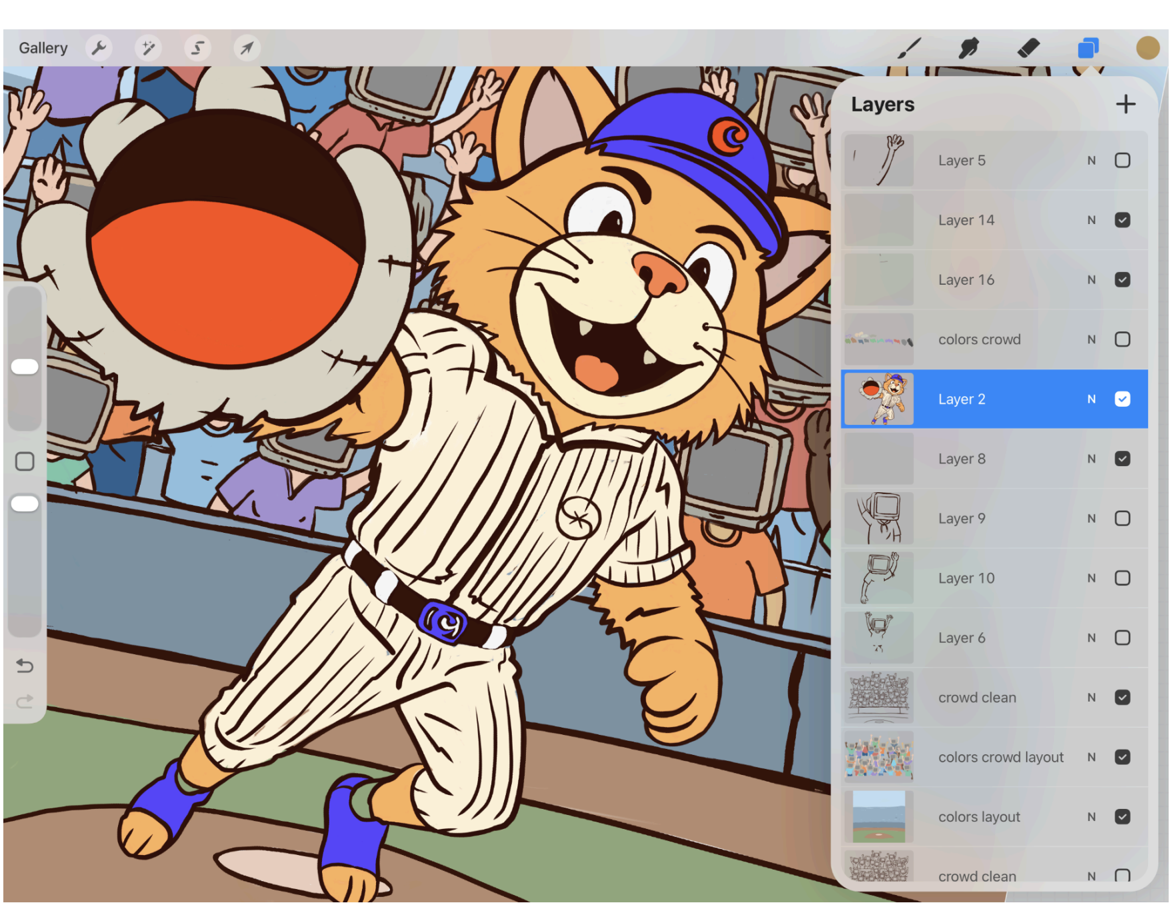
Gallery
Layers
Layer 5
Layer 14
Layer 16
colors crowd
Layer 2
Layer 8
Layer 9
Layer 10
Layer 6
crowd clean
colors crowd layout
colors layout
crowd clean

Figure 11: Capcat Illustration 2

The slogan ties the core functionality in one line:

> "Archive Articles with Confidence. Share without Limits."

Read the full Case Study here:

CAPCAT TUI/CLI - ETHICAL WEB SCRAPER (https://stayux.com/works/capcat-cli/)