PERSONAL REFERENCE · INTERNAL

SKU Skills Universe Architecture

A personal reference for the WAT framework and Skills system — how it's structured, how skills are built, and how everything fits together across projects.

FRAMEWORK	STACK	LAST UPDATED
WAT v1	Claude Code + Python	Feb 2026

01 — THE WAT FRAMEWORK

WAT separates **reasoning** from **execution**. AI handles orchestration; deterministic scripts handle the actual work. This is what keeps accuracy high across multi-step tasks.

W

Workflows

Markdown SOPs in `workflows/`. Define objective, inputs, tools, outputs, and edge cases — written in plain language.

```
workflows/01-capture-app-ui.md
```

A

Agent

Claude Code — the decision-maker. Reads workflows, picks the right tools, handles failures, asks questions, improves the system.

```
claude-code (this session)
```

T

Tools

Python scripts in `tools/` or `scripts/`. One job each. Consistent, testable, reusable across projects.

```
tools/pageflows_capture.py
```

Why it works: If each step is 90% accurate, five chained steps = 59% success. By offloading execution to deterministic scripts, the agent stays focused on reasoning — where it excels.

02 — SKILLS: PORTABLE WAT PACKAGES

Skills are **portable WAT bundles**. Instead of rebuilding tools and workflows per project, skills live in `~/.claude/skills/` and load into any session instantly.

```
# System-wide skills (available in all projects) ~/.claude/skills/ ├── ui-clone/ ← WAT for cloning any app UI ├── claude-ui/ ← Claude.ai 19 flows + all components ├── agent-browser/ ← Browser automation + PageFlows tips ├── ultimateuiux/ ← 93 styles, 121 palettes, design system ├── nextjs/ ← Next.js 16 App Router patterns ├── skill-creator/ ← How to build new skills └── ... 15+ more skills # Per-project context project/ ├── CLAUDE.md ← Project instructions + WAT setup ├── workflows/ ← Project-specific SOPs ├── tools/ ← Project-specific scripts └── .env ← API keys (never in skills)
```

The key difference: project `tools/` are local. Skills are **shared across all projects** and version-controlled on GitHub.

03 — SKILL ANATOMY

Every skill follows the same structure. `SKILL.md` is the entry point — everything else is supplementary.

`SKILL.md`

Main index — loaded on invoke. Must stay under 150 lines.

→ Frontmatter: name + description (<200 chars)
→ Quick-start commands
→ Critical rules (most common bugs)
→ Links to references/ (NOT duplicates)

`references/*.md`

Deep-dive docs. Loaded on demand. Max 150 lines each.

→ Component details, state diagrams
→ Page layouts (ASCII diagrams)
→ Design tokens table
→ Known bugs + fixes

`scripts/*.py`

Deterministic tools — the T in WAT. One job each.

→ Never hardcode credentials
→ Read from .env or args
→ Must be testable standalone
→ requirements.txt alongside

`assets/ (optional)`

Templates, examples, screenshots referenced by refs.

→ No large binary files
→ Screenshots: path refs only
→ Templates: fill-in placeholders

04 — CURRENT SKILLS CATALOG

● **`ui-clone`**
WAT framework for cloning any app from PageFlows. 3-phase: capture → build → package skill.

● **`claude-ui`**
Claude.ai full UI clone. 19 flows, all design tokens, every component. Next.js 16 + Tailwind v4.

● **`agent-browser`**
Browser automation CLI. Core workflow: open → snapshot → interact → re-snapshot. PageFlows tips

● **`nextjs`**
Next.js 16 App Router: use cache, RSC, streaming, PPR, Prisma 7, Better Auth.

included.

`ultimateuiux`

93 styles, 121 palettes, 81 font pairings, 65 WCAG criteria + Claude.ai blueprint.

`skill-creator`

How to build new skills. Templates, validation, packaging workflow.

`auth / api / security`

Quality systems: score → fix → loop until target. OWASP-mapped, Better Auth patterns.

`stripe / payment / email`

Stripe integration, Resend + React Email, billing flows. 7 modes each.

05 — PROJECT → SKILL LIFECYCLE

Every successful project workflow should be packaged as a skill. This is the self-improvement loop made permanent.

1. **New Project** — CLAUDE.md + WAT structure initialized →
2. **Run Workflows** — Invoke relevant skills, execute tools →
3. **Fix + Iterate** — Visual diff loop, self-improvement →
4. **Package Skill** — generate_skill_refs.py → fill refs →
5. **Push GitHub** — Available in all future projects

```
# Phase 1: Bootstrap
python scripts/pageflows_capture.py --app notion
python scripts/find_shared_components.py --manifest .tmp/notion/manifest.json

# Phase 2: Build with visual validation
python scripts/visual_diff.py --original screenshots/notion/home/01.png --url http://localhost:3000
# Target: ≥80% match on every screen

# Phase 3: Package as skill
python scripts/generate_skill_refs.py --app notion --component-map .tmp/notion/component-map.json
cd ~/.claude/skills && git add notion-ui/ && git push
```

06 — QUALITY RULES

RULE	LIMIT	WHY
SKILL.md size	`< 150 lines`	Fits in context window without truncation
Reference file size	`< 150 lines`	Each file loaded on demand — keep focused
Description frontmatter	`< 200 chars`	Used for skill matching and discovery

No duplicate info	`DRY`	Info in SKILL.md OR references, never both
Visual diff threshold	`≥ 80%`	Use --threshold 70 for font-heavy screens
Critical rules minimum	`≥ 5 entries`	From actual implementation failures — most valuable part
Scripts: no hardcoded secrets	`.env only`	Skills are public on GitHub

07 — SELF-IMPROVEMENT LOOP

Every bug found in a future session is an opportunity to make the skill permanently better.

🔍	🔧	✅	📝	🚀
Find Bug	**Fix It**	**Verify**	**Document**	**Push**
Visual diff fails or wrong output	Update component or script	Re-run diff, confirm ≥80%	Update skill ref + Critical Rules	git push → available everywhere