BUILD GUIDE · AGENT-J+

CLAUDE CODE

with memory.

Six markdown files that give Claude Code **persistent context across every session**.
Stop starting blind. Stop re-investigating your own repo.

THE SIX FILES

- TECH_STACK.md
- PARITY_CHECKLIST.md
- DOWNLOAD_FLOWS.md
- TESTING_PROTOCOL.md
- SESSION_CLOSEOUT.md
- INVESTIGATIONS.md

Agent-J+
agent-j-plus.com/docsystem

Claude Code with Memory: A Documentation System for AI-Assisted Builds

Stack Level: Level 3 — Claude Code
Tools: Claude Code, Claude Cowork, CLAUDE.md convention, Markdown
Time to build: \~45 minutes (initial setup), then 5 minutes per session closeout
Difficulty: Intermediate

What You're Building

You run Claude Code sessions on your codebase. Each session starts fresh. Claude reads files, investigates, finds things, fixes things, and by the end of the session, it knows your codebase inside out.

Then the session ends. All that knowledge evaporates.

Next session, you ask Claude to fix a similar issue. It re-investigates the same architecture. Re-discovers the same gotchas. Burns tokens re-learning what it already knew yesterday. Worse: it might miss something it caught last time, because the new session has no memory of the old one.

This build guide sets up a documentation system that solves this. Six structured files that live in your repo, give Claude Code persistent context across sessions, and get updated automatically at the end of every work session.

Here's what the system does:

- Gives Claude Code a reference for your full tech stack on first read (no more "let me scan the codebase to understand...")
1. Enforces a checklist so features work on every surface before they merge
- Documents your data pipelines so Claude doesn't have to reverse-engineer them each time
- Standardizes how tests are written and named
- Runs a mandatory end-of-session routine that updates every doc
- Keeps an append-only investigation log so no finding is ever lost

Your remaining role: review the session closeout, approve commits, decide what gets investigated next. The documentation machine runs itself.

The Chain

Step	Who	What It Does	Hands Off To
1	You (once)	Create the docs/claude/ folder in your repo	→ Claude Code
2	Claude Code (once)	Scan the codebase and seed TECH_STACK.md	→ You (review)
3	You (once)	Define your surfaces/pipelines for the checklist and flow docs	→ Claude Code
4	Claude Code (once)	Generate \`PARITY*CHECKLIST.md, DOWNLOAD*FLOWS.md, TESTING\_PROTOCOL.md\`	→ You (review)
5	You (once)	Add SESSION_CLOSEOUT.md and INVESTIGATIONS.md templates	→ Claude Code
6	Claude Code (every session)	Reads all 6 docs at start, updates them at end via SESSION_CLOSEOUT	→ Next session

Steps 1 through 5 are one-time setup. Step 6 runs every session automatically.

Prerequisites

1. Claude Code (Pro or Max subscription)
1. A codebase with enough complexity that sessions lose context (typically any project with 50+ files, multiple rendering surfaces, or external service integrations)
1. Git version control on the repo
1. Basic familiarity with CLAUDE.md convention (Claude Code reads CLAUDE.md files automatically when they exist in the project)

Zone 1: Folder Structure (Foundation)

Create the documentation home inside your repo so Claude Code finds it automatically.

Step 1: Create the docs folder

In your project root, create:

your-project/

├── docs/

│ └── claude/

│ ├── TECH_STACK.md

│ ├── PARITY_CHECKLIST.md

│ ├── DOWNLOAD_FLOWS.md

│ ├── TESTING_PROTOCOL.md

│ ├── SESSION_CLOSEOUT.md

│ └── INVESTIGATIONS.md

└── CLAUDE.md ← your project's root instructions

Step 2: Wire it into CLAUDE.md

Add this block to your project's root CLAUDE.md so Claude Code reads the documentation at every session start:

\#\# Documentation System

Before starting any task, read these files in order:

1\. docs/claude/TECH_STACK.md — full architecture reference

2\. docs/claude/PARITY_CHECKLIST.md — surfaces to verify for every feature

3\. docs/claude/INVESTIGATIONS.md — historical findings (never repeat work)

Before ending any session, run the closeout routine:

\- docs/claude/SESSION_CLOSEOUT.md — mandatory end-of-session checklist

Setting	Value	Notes
Folder location	docs/claude/	Inside the repo, not external
CLAUDE.md reference	Root-level CLAUDE.md	Claude Code reads this automatically
Read order	TECH_STACK → PARITY → INVESTIGATIONS	Architecture first, then rules, then history

Why these values:

- **Inside the repo:** The documentation travels with the code. When you branch, the docs branch too. When you deploy, the docs are in the same commit. No external files to lose or forget.
- **Root CLAUDE.md reference:** Claude Code reads CLAUDE.md at the start of every session without being asked. By pointing it to the 6 files from there, every session starts with full context.
- **Read order:** Architecture context first (so Claude understands what it's working with), then rules (so it knows what to check), then history (so it doesn't re-investigate old findings).

Zone 2: TECH_STACK.md (Architecture Reference)

The single source of truth for your codebase's architecture. Claude reads this instead of scanning hundreds of files.

Step 3: Seed the tech stack

Run this prompt in Claude Code to generate the initial version:

Scan this entire codebase and create docs/claude/TECH_STACK.md with the following sections:

1\. \\Frontend\\ — Framework, language, bundler, styling approach

2\. \\Backend\\ — Database, auth, API layer, migrations count

3\. \\External Services\\ — Every third-party integration (payment, AI, CDN, etc.)

4\. \\Component Map\\ — Every major directory and what lives in it

5\. \\State Management\\ — How state flows through the app

6\. \\Config Constants\\ — Where pricing, limits, feature flags

live (single source of truth files)

7\. \\Build & Deploy\\ — How the app builds, where it deploys, CI/CD pipeline

8\. \\Key Files\\ — The 10-15 files that matter most (entry points, config, core logic)

Use exact file paths. Include file counts per directory. List every external dependency by name and purpose.

Do not include code snippets longer than 5 lines. This is a reference map, not a code dump.

Save to docs/claude/TECH_STACK.md. STOP when done.

What Claude does: Reads the full project tree, package.json/requirements.txt, config files, and key source files. Produces a structured reference document with exact paths.

Worked example (Aspectr):

The Aspectr TECH_STACK covers: React/TypeScript/Vite frontend, Supabase backend with 64 migrations and 9 edge functions, Transloadit for video processing, Stripe for payments, Gemini AI for smart cropping, and Netlify deployment. Key files section lists config/plans.ts as the single source of truth for pricing.

Step 4: Review and correct

Read through the generated TECH_STACK. Fix anything Claude got wrong. Add context it can't infer: why you chose certain tools, which integrations are legacy, which patterns are intentional vs. accidental. This is the one file where your knowledge matters more than Claude's analysis.

Zone 3: PARITY_CHECKLIST.md (Surface Coverage)

Forces Claude to verify features across every surface before a PR can merge. The number one cause of regression bugs: a feature works in the editor but breaks in the export.

Step 5: Identify your surfaces

Every project has multiple places where the same feature needs to work.

For a web app, this might be:

- Editor view
- Preview view
- Export/download flow
- Mobile view
- API response
- Email rendering
- PDF generation

List every surface where a feature change must be verified.

Worked example (Aspectr):

Aspectr has 7 surfaces for every overlay feature:

\#	Surface	Renderer	Key File
1	Image Editor	Fabric.js canvas	clipCanvasRenderer.ts
2	Video Editor	Fabric.js with ghost styling	clipCanvasRenderer.ts
3	Image Preview	Composite canvas	renderCompositeToCanvas .ts
4	Video Preview	Video processor	videoProcessor.ts
5	Image Download	Canvas API	imageProcessor.ts
6	Video Download	Transloadit assembly	Pre-render to PNG
7	Extracted Frames	Post-processing overlay	renderCompositeToCanvas ()

The rule: a feature PR cannot merge until it works on all 7 surfaces.

Step 6: Create the checklist

Run this prompt in Claude Code:

> *Read docs/claude/TECH_STACK.md to understand the architecture.*
>
> *Create docs/claude/PARITY_CHECKLIST.md with:*
>
> *1\. A numbered list of every surface where visual/functional features must be verified*
>
> *2\. For each surface: the renderer technology, the key file path, and the specific function or entry point where features "drop in"*
>
> *3\. A section called "Drop Points" listing the exact rendering steps where features commonly get lost*

4\. A merge rule: "No PR merges until all surfaces pass"

Use the surfaces I identified: \[paste your surface list here\]

Save to docs/claude/PARITY_CHECKLIST.md. STOP when done.

Why these values:

- **Exact file paths:** Claude needs to know WHERE to implement, not just WHAT to implement. "Image download" is useless. "imageProcessor.ts:compositeOnBlob()" is actionable.
- **Drop points:** These are the specific steps in a rendering pipeline where features get lost. In Aspectr, edge blur must be implemented in 3 separate locations that must stay in sync. Without the drop point documented, Claude will implement it in one location and miss the other two.

Zone 4: DOWNLOAD_FLOWS.md (Pipeline Documentation)

Documents your data pipelines end to end. Any process where data transforms, moves between services, or follows a different path depending on input type.

Step 7: Map your pipelines

If your app has distinct processing paths (image vs. video, sync vs. async, client-side vs. server-side), each one gets documented in DOWNLOAD_FLOWS.

Run this prompt:

Read docs/claude/TECH_STACK.md.

Create docs/claude/DOWNLOAD_FLOWS.md documenting every distinct data/processing pipeline in this app.

For each pipeline:

1\. Name and trigger (what starts it)

2\. Every step in order, with file paths

3\. Where external services are involved (APIs, webhooks, processing queues)

4\. The critical difference between pipelines (what makes them fundamentally different, not just different parameters)

5\. Points where features must stay in sync across pipelines

Use ASCII flow diagrams for each pipeline.

Save to docs/claude/DOWNLOAD_FLOWS.md. STOP when done.

Worked example (Aspectr):

Aspectr has two fundamentally different download pipelines:

IMAGE DOWNLOAD (client-side):

User clicks → imageProcessor.ts:compositeOnBlob() → Canvas API → JSZip → browser download

VIDEO DOWNLOAD (server + external service):

User clicks → pre-render overlays to PNG → Transloadit assembly → webhook → signed URL → browser download

The critical difference: Transloadit only accepts image watermarks. All text, shapes, and effects must be pre-rendered to PNG before submission. This means video download has an extra step that image download doesn't, and any new overlay feature needs to account for that pre-render step.

Zone 5: TESTING_PROTOCOL.md (Test Standards)

How tests are structured, named, and gated.

Step 8: Define your testing convention

Read docs/claude/PARITY_CHECKLIST.md.

Create docs/claude/TESTING_PROTOCOL.md with:

1\. Test framework and runner (Playwright, Jest, Vitest, etc.)

2\. Naming convention: one test file per surface per feature

Pattern: `\[feature-name\].\[surface\].spec.ts\`

3\. Merge gate: all surface tests must pass before PR

4\. When tests are required: any change to \[your trigger criteria\]

5\. Test structure template (describe → it → assert pattern)

Reference the surfaces from PARITY_CHECKLIST.md by name.

Save to docs/claude/TESTING_PROTOCOL.md. STOP when done.

Setting	Value	Notes
Naming convention	\[feature\].\[surface\].spec.ts	One file per surface per feature
Merge gate	All surface tests pass	Enforced in PARITY_CHECKLIST
Trigger	Changes to overlay rendering, core components, or shared utilities	Adapt to your codebase

Why these values:

- **One file per surface per feature:** When a test fails, you immediately know which surface broke. No digging through a 500-line test file to find the failure.
- **Merge gate:** Without this, the parity checklist becomes a suggestion. The gate makes it a requirement.

Zone 6: SESSION_CLOSEOUT.md (The Engine)

The mandatory end-of-session routine. This is what makes the system self-sustaining. Without it, the other 5 files go stale within a week.

Step 9: Create the closeout routine

This file is a checklist that Claude Code runs at the end of every session. Create it manually (this one is too important to auto-generate):

\# Session Closeout — Mandatory

Run this checklist before ending any Claude Code session on this

project.

This is non-negotiable. Skipping it means the next session starts blind.

\#\# 1. Diff Summary

Write a one-paragraph summary of what changed in this session and why.

List every file modified with a one-line explanation.

\#\# 2. Doc Updates

Check each documentation file and update if anything changed:

\- \[\] TECH_STACK.md — new dependencies, changed architecture, new key files?

\- \[\] PARITY_CHECKLIST.md — new surfaces, changed file paths, new drop points?

\- \[\] DOWNLOAD_FLOWS.md — pipeline changes, new external services, sync point changes?

\- \[\] TESTING_PROTOCOL.md — new test patterns, changed naming, new trigger criteria?

\#\# 3. Backlog Update

\- Mark completed tickets/issues as done

\- Add any new issues discovered during the session

\#\# 4. Test Update

If a visual or functional feature was added/changed:

\- Add parity tests per TESTING_PROTOCOL.md

\- Run existing tests to confirm no regressions

\#\# 5. Investigation Log

If any non-trivial finding was made during this session:

\- Append to INVESTIGATIONS.md (see format below)

\- Include: date, symptom, root cause, fix, lesson

\#\# 6. Commit Message

Include a session closeout summary in the commit message:

"Session closeout: \[one-line summary of what was done and documented\]"

Step 10: Create the investigation log

Create INVESTIGATIONS.md as an append-only file:

\# Investigation Log

Append-only. Never delete old entries. Newest entries at the top.

Format: Date — One-line summary, then details.

\---

\[Entries will be added by SESSION_CLOSEOUT routine\]

The format for each entry:

\#\# 2026-04-16 — \[One-line summary\]

\\Symptom:\\ \[What was observed\]

Root cause:\\ *[What actually caused it\]*

Fix:\\ *[What was changed, with file paths\]*

Lesson:\\ *[What to remember for next time\]*

Why this matters:

In the Aspectr project, we burned 2 full days investigating a text caret/selection bug. The root cause was a setTimeout(selectAll) in a double-click handler. Without the investigation log, the next time someone reports a selection issue, Claude would start the investigation from scratch: wrong theories, wrong files, wrong approaches.

With the log, Claude reads the entry and knows: "Selection bugs in this codebase, check the double-click handlers for setTimeout calls first."

That's not a 1-4% token saving. That's an entire investigation you never have to run again.

Activate

Session Flow

When	What Happens
Session start	Claude Code reads CLAUDE.md → reads all 6 docs → has full context
During work	Claude references TECH*STACK for architecture, PARITY*CHECKLIST for coverage, DOWNLOAD_FLOWS for pipeline details
Before PR	Claude checks PARITY*CHECKLIST surfaces, runs tests per TESTING*PROTOCOL
Session end	Claude runs SESSION_CLOSEOUT → updates all docs → appends to INVESTIGATIONS

Go-Live Checklist

- \[\] docs/claude/ folder exists in your repo
- \[\] All 6 files created and populated
- \[\] Root CLAUDE.md references the 6 files with read-order instructions
- \[\] TECH_STACK.md reviewed and corrected by you (the human)
- \[\] PARITY_CHECKLIST surfaces match your actual rendering/output surfaces
- \[\] First Claude Code session started, docs read confirmed,

session closeout completed

Manual Testing

1. Start a new Claude Code session on your project
1. Ask Claude: "What framework does this project use?" — it should answer from TECH_STACK without scanning files
1. Ask Claude: "What surfaces need to be checked for a new overlay feature?" — it should list them from PARITY_CHECKLIST
1. Make a small change, then say "run the session closeout"
1. Check that INVESTIGATIONS.md has a new entry (if anything was investigated)
1. Check that the commit message includes the closeout summary

Known Limitations

Limitation	Impact	Workaround
TECH_STACK goes stale if major refactors happen outside Claude Code sessions	Claude works from outdated architecture	Re-run the seed prompt after major refactors
INVESTIGATIONS.md grows forever	File gets long over months	Archive entries older than 6 months to INVESTIGATIONS_ARCHIVE.md
Claude sometimes skips the closeout if the session ends abruptly	Docs don't get updated	Add "Always run SESSION_CLOSEOUT before your final commit" to CLAUDE.md
Parity checklist is manual to maintain	New surfaces might get missed	Review the checklist quarterly or when adding new output formats

On Token Savings

Here's the honest answer: we don't have hard numbers yet.

What we know from the RTK/Caveman investigation (see the companion post): tools that promise 60-90% token savings by filtering shell output deliver 1-4% in real-world Claude Code sessions. Output filtering is a dead end for meaningful savings.

What we believe, but haven't measured: the real waste in Claude Code sessions is re-investigation. Re-scanning the same architecture. Re-discovering the same gotchas. Missing a surface because the session didn't know about it, then spending another session fixing the regression.

The documentation system attacks that waste directly. Instead of filtering output, it prevents the work from happening in the first

place.

We're tracking this. Once we have a month of session data with the system running, we'll publish what we find. If the savings are 5%, we'll say 5%. If they're 40%, we'll say 40%. No inflated promises.

The principle: measure first, claim later. The opposite of what we found with RTK.

Grab & Go

CLAUDE.md Convention — Claude Code reads this file automatically at session start. Put your documentation references here.

SESSION_CLOSEOUT template — Copy the template from Zone 6 and adapt the checklist items to your project.

Investigation Log format — The Symptom/Root Cause/Fix/Lesson structure works for any codebase. Start using it even before setting up the full system.

File Structure (After Build)

your-project/

├── docs/

│ └── claude/

│ ├── TECH_STACK.md ← Architecture reference (seeded by Claude, reviewed by you)

│ ├── PARITY_CHECKLIST.md ← Surface coverage rules

│ ├── DOWNLOAD_FLOWS.md ← Pipeline documentation

│ ├── TESTING_PROTOCOL.md ← Test naming and gating rules

│ ├── SESSION_CLOSEOUT.md ← End-of-session routine

│ └── INVESTIGATIONS.md ← Append-only finding log

├── CLAUDE.md ← Root instructions pointing to docs/claude/

└── *\[your source code\]*

Costs

What	Cost
Claude Code (Pro or Max)	$20 to $100/month (you're already paying this)
Additional token cost for closeout routine	\~2-5 minutes per session (minimal)
Total additional cost	**\~$0**

The system adds a few minutes of token usage per session for the closeout. It removes entire sessions worth of re-investigation. The math works in your favour even before we have exact numbers.
Claude Unlocked · Stack Level 3 · Built by Balazs · AI Automation School