The Prompt to Personalize Your Second Brain Build

The architecture is the same for everyone — capture, classify, file, confirm, digest, fix. But the stack that actually ships depends on where you already live, what's falling through the cracks, and how much you're willing to maintain. These four prompts force you to answer those questions before you build and give you a personalized build guide so you don't waste a weekend on a system that doesn't fit your life.

This kit will not recommend tools until it understands your purpose and constraints. If you leave blanks, it will ask questions and stop. Do not skip ahead.

Prompt 1: What should your second brain actually do?

You are helping me design a second brain — but before we pick tools, I need to know what it's actually FOR.

RULES:

- If any answer below is blank or vague, ask up to 6 clarifying questions (all at once) and STOP. Do not proceed until inputs are complete.

- Do not assume defaults. Do not infer what I "probably" mean.

- Outputs must be observable artifacts (digest, DM, list, dashboard, weekly note) — not feelings.

MY SITUATION:

1. What's falling through the cracks right now: ___

2. What I've tried before and why it failed: ___

3. Where friction kills me most (capture / retrieval / follow-through / all three): ___

4. If this works, what do I stop doing every week? ___

5. What would "working" look like — what do I want to receive or see? ___

6. What information types am I capturing most? (pick 2–3): meetings / personal life / work tasks / research / relationship notes / creative ideas / admin errands

7. What's the cost of being wrong? (if it misfiles or loses something): low / medium / high

OUTPUT (only after inputs are complete):

1. Your primary use case (one sentence: "A system that helps you ___")

2. The 3 observable outputs that would make this worth building (be specific: "a weekly DM listing people you haven't followed up with" not "reminders")

3. The 2 things this system should NOT try to do (scope control)

4. The operating mode that fits how you actually work (always-on / scheduled / session-based) + why

5. The reliability tier this requires (based on cost-of-wrong): "good enough" / "needs receipts" / "full audit trail"

6. The one weekly task this replaces if it works

Prompt 2: What stack fits your reality?

Based on what I want my second brain to do, recommend the right tool stack.

RULES:

- If any required field is blank, ask all missing questions (max 8) and STOP. Do not recommend anything until inputs are complete.

- Do not assume defaults. No "I'll assume you use Slack."

- Only recommend tools from MY allowed list. Do not suggest tools I've refused.

- Recommend ONE primary stack. Offer ONE simpler fallback only if meaningfully different.

MY SECOND BRAIN PURPOSE (paste from Prompt 1 or fill in):

- Primary use case: ___

- Key outputs I need: ___

- Operating mode (always-on / scheduled / session-based): ___

- Reliability tier (good enough / needs receipts / full audit trail): ___

MY REALITY:

- Where I already live (communication — Slack / Discord / Email / iMessage / Teams / other): ___

- Where I already live (notes/docs — Notion / Obsidian / Google Docs / Apple Notes / other): ___

- Automation I already use (Zapier / Make / n8n / none / other): ___

- AI access (ChatGPT / Claude / both / work account only / none): ___

- Technical comfort: non-technical / light automation / comfortable with code

- Time for setup: 30 min / 2 hours / weekend project

- Budget: $0 / <$20/mo / doesn't matter

- Devices (iPhone / Android / Mac / Windows — list all): ___

- Privacy/compliance: none / prefer local / work policies / HIPAA-PHI / other: ___

- Work constraints: can you install apps / connect APIs / use personal accounts at work? ___

- Captures per day (volume): 1–5 / 5–20 / 20+

- Latency tolerance: instant / within an hour / once a day / only when I run it

- Where capture happens most: phone / desktop / both

- Tools I refuse to use: ___

- Tools I'm willing to add (max 1 new tool): ___

OUTPUT (only after inputs are complete):

A) Recommended stack mapped to your purpose:

  - Capture:

  - Automation:

  - Storage:

  - AI:

  - Delivery:

B) Why this stack fits your operating mode, volume, and reliability tier

C) The simplest fallback stack (if meaningfully different)

D) What you lose with the fallback (be specific)

E) The one new tool you'll need to learn + why it's worth it (or "none")

F) The highest-risk dependency in this stack (vendor / permission / cost / brittleness)

Prompt 3: Build my personalized guide

Generate a personalized, runnable build guide for my second brain.

RULES:

- If any required field is blank, ask all missing questions (max 8) and STOP. Do not generate a guide until inputs are complete.

- Do not assume defaults.

- Constrain to: max 4 destination buckets (people / projects / ideas / admin) + inbox log. Max 5 required fields per bucket. Everything else is optional later.

- If user is non-technical, provide ONLY no-code approaches. For parsing: use JSON mode / response_format if available, formatter steps to strip markdown, or fallback to Needs Review if parsing fails. Specify exactly where each fallback goes in the automation.

- Output must end in a testable, closed loop — see DEFINITION OF DONE below.

MY SETUP:

- Purpose: ___

- Stack: Capture (___) / Automation (___) / Storage (___) / AI (___) / Delivery (___)

- Operating mode (always-on / scheduled / session-based): ___

- Technical comfort: non-technical / light automation / comfortable with code

- Reliability tier (good enough / needs receipts / full audit trail): ___

DEFINITION OF DONE (the guide must close this loop):

✓ A capture creates a record in the correct destination bucket

✓ An inbox log entry is created with a pointer back to the original capture

✓ A confirmation message is sent to the capture channel

✓ A low-confidence path exists (needs review bucket or flag)

✓ A fix flow updates the log and refiles to the correct destination

✓ A digest runs on schedule OR a session checklist exists (depending on operating mode)

OUTPUT FORMAT:

0. MVP BUILD (30 minutes — do this first, stop here on day 1)

- Build ONLY: capture → classify → file → confirm → inbox log

- Skip for now: fix flow, digests (add on day 2)

- The exact test message to send: ___

- The expected result (which database, which fields, which confirmation): ___

- How to know it worked: ___

1. SETUP PHASE (do once)

- Exactly what to create in each tool — name the channels, databases, folders

- Database schema for each bucket (max 5 required fields each — name them)

- The connections to wire (which trigger → which action)

- The full classification prompt in the exact schema your storage layer needs

- Confidence threshold to use
- Fix command format (e.g., "fix: people")
- System of record: which tool is authoritative (Storage vs Inbox Log vs Capture channel)
- Conflict rule: if the same item exists in two places, which version wins?

2. COPY/PASTE ASSETS

- Channel description / pin text for capture channel
- Database property names (exact casing, exact spacing)
- Classification prompt (full, copy-ready)
- Digest prompt (full, copy-ready)
- Parsing fallback plan:
  - If using code: the exact code step
  - If non-technical: which formatter step or JSON mode setting to use, and where
  - If all parsing fails: route to Needs Review with original text preserved

3. TEST PHASE (before you trust it)

- 5 test messages that cover: simple capture, ambiguous item, person mention, project with deadline, and a "fix" correction
- What to check after each one (which database, which fields, which confirmation)
- The most likely failure point and how to diagnose it

4. OPERATING RULES (enforceable, not advice)

- One capture rule (e.g., "one message = one thought")
- One fix rule (e.g., "fix in-thread within 24h or ignored")
- One maintenance rule (e.g., "clear Needs Review weekly, max 5 minutes")

5. EXTENSION IDEAS (later, not now)

- 2 things to add once core loop works for 30 days

- 1 thing to explicitly NOT add yet

Format as a checklist. No prose. Every item must be an action or a decision.

Prompt 4: Stress test before you build

Stress-test my second brain plan before I build it.

RULES:

- If the plan below is incomplete or vague, ask up to 5 clarifying questions and STOP.

- Do not invent details about my setup.

MY PLAN:

- Purpose: ___

- Stack: Capture (___) / Automation (___) / Storage (___) / AI (___) / Delivery (___)

- Operating mode: ___

- Volume (captures/day): ___

- Technical comfort: ___

- Reliability tier: ___

TASKS:

1. List the 5 most likely ways this breaks in real life (not theoretical edge cases):

   For each:

- How I'll notice it
- The fastest fix
- One enforceable rule that prevents it

2. Which single component is the highest-risk dependency? (vendor lock-in / permission fragility / cost spike / brittleness)
   - What's the fallback if it fails?

3. Should I simplify this plan? (yes / no)
   - If yes: what to cut and what simpler mode to use
   - If no: what's the one thing I must not cut

4. The one habit that will make or break this system (not "be consistent" — something I can put in my calendar or automate)

5. Day-1 test plan:

   FAILURE-INJECTION TESTS (3):
   - Malformed AI response (JSON wrapped in markdown or with preamble) — what should happen?
   - Missing permission or disconnected integration — how will you know?
   - Low-confidence classification (ambiguous input) — where does it go?

   NORMAL TESTS (2):
   - Simple capture → file → confirm (verify the happy path)
   - Fix correction → refile → updated log (verify the repair loop)

For each: what should happen, and how to verify it did.

Quick Reference: The Flow

Step	Prompt	Output
1	What should it do?	Purpose, outputs, operating mode, reliability tier
2	What stack fits?	Tools, fallback, risks
3	Build guide	MVP checklist, full setup, copy/paste assets
4	Stress test	Failure modes, Day-1 tests, habit anchor

Do not skip to Prompt 3. The inputs from 1 and 2 prevent the guide from being generic.

Patch Notes (what makes this bulletproof)

Failure Mode	Prevention
Model infers when inputs missing	"Ask and STOP" gate on every prompt
Guide doesn't close the loop	Explicit Definition of Done checklist
Wrong automation for volume	Captures/day + latency tolerance inputs
Recommends tools user hates	"Tools I refuse" + "max 1 new tool" fields
People overbuild databases	Max 4 buckets, max 5 fields constraint
Non-technical user gets code steps	Explicit no-code parsing fallback requirement
People quit before loop works	MVP BUILD (30 min) section comes first
Habits section is motivation	Renamed to Operating Rules — enforceable
No system of record	Added authoritative tool + conflict rule
Outputs too abstract	Forced observable artifacts

Failure Mode	Prevention
"Cost of wrong" ignored	Added reliability tier (good enough → full audit)
Day-1 tests redundant	Split into failure-injection vs normal tests