Second Brain Build Guide

A Complete No-Code Implementation with Slack, Notion, Zapier, and Claude/ChatGPT

Overview

This system has three Zapier automations, five Notion databases, and one Slack channel. Total build time is 1-2 hours.

The Core Loop

1. You capture a thought in Slack (5 seconds)
2. Zapier sends it to Claude/ChatGPT for classification
3. The AI returns structured JSON with category, fields, and confidence
4. Zapier routes it to the correct Notion database
5. Zapier replies in Slack confirming what it did
6. Daily/weekly digests surface what matters

The Three Automations

- **Capture Zap:** Slack message → AI classification → Notion filing → Slack confirmation
- **Fix Zap:** Slack reply containing "fix" → Parse correction → Update Notion record
- **Digest Zap (x2):** Scheduled trigger → Query Notion → AI summarization → Slack DM

Prerequisites

Required Accounts

- Slack (free tier works)
- Notion (free tier works)
- Zapier (free tier allows 5 zaps with 100 tasks/month; you'll likely need a paid plan for volume)
- OpenAI API key OR Anthropic API key (for Claude)

Estimated Costs

Service	Cost
Zapier Starter	~$20/month
OpenAI API	~$5-15/month (GPT-4o-mini is cheapest)
Anthropic API	~$5-15/month (Claude 3.5 Haiku is cheapest)

Alternative: Use Make.com instead of Zapier for lower costs (~$9/month).

Part 1: Slack Setup

Step 1.1: Create Your Capture Channel

1. Open Slack
2. Click the + next to "Channels"
3. Create a new channel:
   - **Name:** `sb-inbox` (or whatever you prefer)
   - **Visibility:** Private (recommended)
   - **Description:** "Drop thoughts here. One message per thought. No organizing needed."

Step 1.2: Pin the Instructions

Post this message and pin it:

📥 SECOND BRAIN INBOX

Drop any thought here. One message = one thought.

No need to organize, tag, or structure anything.
The system will classify and file it automatically.

Examples:
- "Sarah mentioned she's looking for a new job"
- "Project: need to finish the Q1 report by Friday"
- "Idea: what if we added a dark mode to the app"
- "Remember to renew car registration"

After filing, I'll reply with what I did. Reply "fix: [correction]" if I got it wrong.

Step 1.3: Note Your Channel ID

1. Right-click on the channel name
2. Click "View channel details"
3. Scroll to the bottom — copy the Channel ID (starts with `C`)

Part 2: Notion Database Setup

Step 2.1: Create the Second Brain Page

1. Open Notion
2. Create a new page called "Second Brain"
3. Inside this page, you'll create 5 databases as inline tables

Step 2.2: Create the People Database

Create a new inline database (type `/database` and select "Table - Inline")

Database Name: People

Property Name	Type	Notes
Name	Title	Person's name
Context	Text	How you know them, their role, relationship
Follow-ups	Text	Things to remember for next conversation
Last Touched	Date	Auto-filled by automation
Tags	Multi-select	Optional: family, work, friend, etc.

Step 2.3: Create the Projects Database

Database Name: Projects

Property Name	Type	Options/Notes
Name	Title	Project name
Status	Select	Active, Waiting, Blocked, Someday, Done
Next Action	Text	The literal next thing to do
Notes	Text	Context, links, details
Last Touched	Date	Auto-filled by automation
Tags	Multi-select	Optional categorization

Step 2.4: Create the Ideas Database

Database Name: Ideas

Property Name	Type	Notes
Name	Title	Idea title
One-Liner	Text	Core insight in one sentence
Notes	Text	Elaboration, related thoughts
Last Touched	Date	Auto-filled by automation
Tags	Multi-select	Optional categorization

Step 2.5: Create the Admin Database

Database Name: Admin

Property Name	Type	Options/Notes
Name	Title	Task name
Due Date	Date	When it needs to be done
Status	Select	Todo, Done
Notes	Text	Additional context
Created	Date	Auto-filled by automation

Step 2.6: Create the Inbox Log Database

This is your audit trail — every capture gets logged here.

Database Name: Inbox Log

Property Name	Type	Notes
Original Text	Title	Exactly what you typed in Slack
Filed To	Select	People, Projects, Ideas, Admin, Needs Review
Destination Name	Text	The name of the created record
Destination URL	URL	Link to the Notion record
Confidence	Number	0-1 score from AI
Status	Select	Filed, Needs Review, Fixed
Created	Date	When it was captured
Slack Thread TS	Text	For linking back to Slack (fix button)

Notion Record ID	Text	For updates (fix button)

Step 2.7: Get Your Database IDs

For each database, you need the ID:

1. Open the database as a full page
2. Look at the URL: `notion.so/yourworkspace/DATABASE_ID?v=...`
3. The DATABASE_ID is the 32-character string before the `?`
4. Save all 5 database IDs somewhere

Part 3: Connect Zapier to Your Tools

Step 3.1: Connect Slack

1. Go to https://zapier.com and log in
2. Go to My Apps (left sidebar)
3. Click Add Connection → Search for "Slack"
4. Authorize Zapier to access your Slack workspace
5. Make sure to select the workspace where your `sb-inbox` channel lives

Step 3.2: Connect Notion

1. In My Apps, click Add Connection → Search for "Notion"
2. Authorize Zapier
3. **Important:** When Notion asks which pages to share, select your "Second Brain" page and all databases inside it

Step 3.3: Connect OpenAI or Anthropic

For OpenAI:

1. Go to https://platform.openai.com/api-keys
2. Create a new API key
3. In Zapier My Apps → Add Connection → "OpenAI"
4. Paste your API key

For Anthropic (Claude):

1. Go to https://console.anthropic.com/settings/keys

2. Create a new API key
3. In Zapier My Apps → Add Connection → "Anthropic (Claude)"
4. Paste your API key

Part 4: Build the Main Capture Automation

This is the core automation that processes every thought you capture.

Step 4.1: Create a New Zap

1. Click Create → New Zap
2. Name it: "Second Brain - Capture"

Step 4.2: Set Up the Trigger

App: Slack
Event: New Message Posted to Channel

Configuration:

- Channel: Select your `sb-inbox` channel
- Trigger for Bot Messages: No
- Trigger for Threaded Replies: No (important — we handle these separately)

Test the trigger to make sure it's working.

Step 4.3: Add the AI Classification Step

App: OpenAI (or Anthropic/Claude)
Event: Send Prompt

For OpenAI:

- Model: `gpt-4o-mini` (cheapest) or `gpt-4o` (better quality)
- User Message: See the Classification Prompt below
- Response Format: JSON Object

For Claude:

- Model: `claude-3-5-haiku-20241022` (cheapest) or `claude-3-5-sonnet-20241022`
- Message: See the Classification Prompt below

The Classification Prompt

Copy this entire prompt into the "User Message" field. Replace `{{Message Text}}` with the dynamic data from your Slack trigger:

You are a classification system for a personal knowledge management system. Your job is to analyze the user's captured thought and return structured JSON.

INPUT:
{{Message Text}}

INSTRUCTIONS:
1. Determine which category this belongs to:
   - "people" - information about a person, relationship update, something someone said
   - "projects" - a project, task with multiple steps, ongoing work
   - "ideas" - a thought, insight, concept, something to explore later
   - "admin" - a simple errand, one-off task, something with a due date

2. Extract the relevant fields based on category

3. Assign a confidence score (0.0 to 1.0):
   - 0.9-1.0: Very clear category, obvious classification
   - 0.7-0.89: Fairly confident, good match
   - 0.5-0.69: Uncertain, could be multiple categories
   - Below 0.5: Very unclear, needs human review

4. If confidence is below 0.6, set destination to "needs_review"

OUTPUT FORMAT (return ONLY this JSON, no other text):

For PEOPLE:
```
{
  "destination": "people",
  "confidence": 0.85,
  "data": {
    "name": "Person's Name",
    "context": "How you know them or their role",
    "follow_ups": "Things to remember for next time",
    "tags": ["work", "friend"]
  }
}
```

For PROJECTS:
```
{
```

```
  "destination": "projects",
  "confidence": 0.85,
  "data": {
    "name": "Project Name",
    "status": "active",
    "next_action": "Specific next action to take",
    "notes": "Additional context",
    "tags": ["work"]
  }
}

For IDEAS:
{
  "destination": "ideas",
  "confidence": 0.85,
  "data": {
    "name": "Idea Title",
    "one_liner": "Core insight in one sentence",
    "notes": "Elaboration if provided",
    "tags": ["product"]
  }
}

For ADMIN:
{
  "destination": "admin",
  "confidence": 0.85,
  "data": {
    "name": "Task name",
    "due_date": "2024-01-15 or null if not specified",
    "notes": "Additional context"
  }
}

For UNCLEAR (confidence below 0.6):
{
  "destination": "needs_review",
  "confidence": 0.45,
  "data": {
    "original_text": "The original message",
    "possible_categories": ["projects", "admin"],
    "reason": "Could be a project or a simple task"
  }
}
```

RULES:
- "next_action" must be specific and executable. "Work on website" is bad. "Email Sarah to confirm deadline" is good.
- If a person's name is mentioned, consider if this is really about that person or about a project/task involving them
- Status options for projects: "active", "waiting", "blocked", "someday"
- Extract dates when mentioned and format as YYYY-MM-DD
- If no clear tags apply, use an empty array []
- Always return valid JSON with no markdown formatting

Step 4.4: Add a Code Step to Parse JSON

Sometimes AI returns JSON wrapped in markdown. This step cleans it up.

App: Code by Zapier
Event: Run Javascript

Input Data:

- `ai_response`: (map to the AI response from previous step)

Code:

```
let response = inputData.ai_response;

// Remove markdown code blocks if present
response = response.replace(/```json\n?/g, '').replace(/```\n?/g, '').trim();

// Parse JSON
try {
  const parsed = JSON.parse(response);
  return {
    destination: parsed.destination,
    confidence: parsed.confidence,
    data: JSON.stringify(parsed.data),
    name: parsed.data.name || parsed.data.original_text || 'Untitled',
    status: parsed.data.status || null,
    next_action: parsed.data.next_action || null,
    context: parsed.data.context || null,
    follow_ups: parsed.data.follow_ups || null,
    one_liner: parsed.data.one_liner || null,
    notes: parsed.data.notes || null,
    due_date: parsed.data.due_date || null,
```

```
    tags: parsed.data.tags ? parsed.data.tags.join(', ') : ''
  };
} catch (e) {
  return {
    destination: 'needs_review',
    confidence: 0,
    data: response,
    name: 'Parse Error',
    error: e.message
  };
}
```

Step 4.5: Add Paths for Routing

App: Paths (built into Zapier)

Create 5 paths:

Path	Conditions
Path A: People	`destination` exactly matches `people` AND `confidence` greater than 0.59
Path B: Projects	`destination` exactly matches `projects` AND `confidence` greater than 0.59
Path C: Ideas	`destination` exactly matches `ideas` AND `confidence` greater than 0.59
Path D: Admin	`destination` exactly matches `admin` AND `confidence` greater than 0.59
Path E: Needs Review	`confidence` less than 0.6 OR `destination` exactly matches `needs_review`

Step 4.6: Set Up Each Path

For each path (A through D), add these actions:

Action 1: Create Notion Database Item

For People Path:

- App: Notion
- Event: Create Database Item
- Database: People
- Properties:
    - Name: `{{name}}` from Code step
    - Context: `{{context}}` from Code step
    - Follow-ups: `{{follow_ups}}` from Code step
    - Last Touched: Current date/time
    - Tags: `{{tags}}` from Code step

For Projects Path:

- Database: Projects
- Properties: Name, Status, Next Action, Notes, Last Touched, Tags

For Ideas Path:

- Database: Ideas
- Properties: Name, One-Liner, Notes, Last Touched, Tags

For Admin Path:

- Database: Admin
- Properties: Name, Due Date, Status (set to "Todo"), Notes, Created

Action 2: Create Inbox Log Entry

After each Notion create action, add another Notion action:

- App: Notion
- Event: Create Database Item
- Database: Inbox Log
- Properties:
    - Original Text: `{{Message Text}}` from Slack trigger
    - Filed To: The category (People/Projects/Ideas/Admin)
    - Destination Name: `{{name}}` from Code step
    - Destination URL: `{{URL}}` from the Notion create action
    - Confidence: `{{confidence}}` from Code step
    - Status: Filed
    - Created: Current date/time
    - Slack Thread TS: `{{Message TS}}` from Slack trigger
    - Notion Record ID: `{{ID}}` from the Notion create action

Action 3: Reply in Slack Thread

- App: Slack
- Event: Send Channel Message
- Channel: Your `sb-inbox` channel
- Thread TS: `{{Message TS}}` from Slack trigger (this makes it a reply)

Message Text:

✅ Filed as {{destination}}

{{name}}
Confidence: {{confidence}}

Reply "fix: [your correction]" if this is wrong.
Example: "fix: this should be people, not projects"

Step 4.7: Set Up the Needs Review Path

Action 1: Create Inbox Log Entry

- Database: Inbox Log
- Filed To: Needs Review
- Destination Name: "Pending Classification"
- Status: Needs Review

Action 2: Reply in Slack

🤔 I'm not sure how to classify this (confidence: {{confidence}})

Could you repost with a prefix?
- "person: ..." for people
- "project: ..." for projects
- "idea: ..." for ideas
- "admin: ..." for tasks/errands

Or reply "fix: [category]" to classify this one.

Step 4.8: Test and Turn On

1. Test each path by sending sample messages to your Slack channel
2. Verify records appear in the correct Notion databases
3. Check that Inbox Log entries are created

4. Turn on the Zap

Part 5: Build the Fix Button Automation

This automation lets you correct mistakes by replying "fix: ..." in Slack.

Step 5.1: Create a New Zap

Name it: "Second Brain - Fix Button"

Step 5.2: Set Up the Trigger

App: Slack
Event: New Message Posted to Channel

Configuration:

- Channel: Your `sb-inbox` channel
- Trigger for Bot Messages: No
- Trigger for Threaded Replies: **Yes** (this is different from the main Zap)

Step 5.3: Add a Filter

App: Filter by Zapier

Condition: `{{Message Text}}` (Text) Contains `fix:`

Step 5.4: Add a Code Step to Parse the Correction

App: Code by Zapier
Event: Run Javascript

Input Data:

- `message`: Map to `{{Message Text}}` from Slack
- `thread_ts`: Map to `{{Thread TS}}` from Slack

Code:

```
const message = inputData.message.toLowerCase();
const threadTs = inputData.thread_ts;

// Extract the correction after "fix:"
```

```
const fixMatch = message.match(/fix:\s*(.+)/i);
if (!fixMatch) {
  return { error: 'Could not parse fix command', valid: false };
}

const correction = fixMatch[1].trim();

// Determine the new destination
let newDestination = null;
const destinations = {
  'people': ['people', 'person'],
  'projects': ['projects', 'project'],
  'ideas': ['ideas', 'idea'],
  'admin': ['admin', 'task', 'errand']
};

for (const [dest, keywords] of Object.entries(destinations)) {
  for (const keyword of keywords) {
    if (correction.includes(keyword)) {
      newDestination = dest;
      break;
    }
  }
  if (newDestination) break;
}

return {
  valid: newDestination !== null,
  new_destination: newDestination,
  correction_text: correction,
  thread_ts: threadTs
};
```

Step 5.5: Add a Filter for Valid Corrections

Condition: `{{valid}}` (Boolean) is true

Step 5.6: Find the Original Inbox Log Entry

App: Notion
Event: Find Database Item

- Database: Inbox Log

- Filter: Slack Thread TS equals `{{thread_ts}}` (the parent thread TS)

Step 5.7: Add Paths Based on New Destination

Create paths for each possible destination (people, projects, ideas, admin).

For each path:

1. **Get the original text** from Inbox Log
2. **Re-run classification** with forced category
3. **Parse the JSON** (same code step as main Zap)
4. **Create record** in the correct Notion database
5. **Update the Inbox Log entry** with new Filed To, Destination Name, Destination URL, and Status: Fixed
6. **Archive the old incorrect record** (optional)
7. **Reply confirming the fix**

Forced Re-classification Prompt
Extract structured data from this text for a {{new_destination}} record.

TEXT:
{{Original Text from Inbox Log}}

CATEGORY: {{new_destination}}

Return ONLY JSON matching this structure:

For "people":
{"name": "...", "context": "...", "follow_ups": "...", "tags": []}

For "projects":
{"name": "...", "status": "active", "next_action": "...", "notes": "...", "tags": []}

For "ideas":
{"name": "...", "one_liner": "...", "notes": "...", "tags": []}

For "admin":
{"name": "...", "due_date": null, "notes": "..."}

Step 5.8: Handle Invalid Fix Commands

Add a final path for when `valid` is false:

❓ I couldn't understand that fix command.

Try: "fix: this should be people" or "fix: move to projects"

Valid categories: people, projects, ideas, admin

Part 6: Build the Daily Digest Automation

Step 6.1: Create a New Zap

Name it: "Second Brain - Daily Digest"

Step 6.2: Set Up the Trigger

App: Schedule by Zapier
Event: Every Day

- Time of Day: Choose when you want your digest (e.g., 7:00 AM)
- Timezone: Your timezone

Step 6.3: Query Active Projects

App: Notion
Event: Find Database Items (plural — finds multiple)

- Database: Projects
- Filter: Status equals "active"
- Limit: 20

Step 6.4: Query People with Follow-ups

- Database: People
- Filter: Follow-ups is not empty
- Limit: 10

Step 6.5: Query Today's Admin Tasks

- Database: Admin
- Filter: Due Date is today OR Due Date is in the past
- AND: Status equals "Todo"
- Limit: 10

Step 6.6: Format the Data for AI

App: Code by Zapier
Event: Run Javascript

Input Data:

- `projects`: JSON string of projects from step 3
- `people`: JSON string of people from step 4
- `admin`: JSON string of admin tasks from step 5

```
const projects = JSON.parse(inputData.projects || '[]');
const people = JSON.parse(inputData.people || '[]');
const admin = JSON.parse(inputData.admin || '[]');

let context = "ACTIVE PROJECTS:\n";
projects.forEach((p, i) => {
  context += `${i+1}. ${p.properties?.Name?.title?.[0]?.plain_text || 'Untitled'}\n`;
  context += `   Status: ${p.properties?.Status?.select?.name || 'Unknown'}\n`;
  context += `   Next Action: ${p.properties?.['Next Action']?.rich_text?.[0]?.plain_text || 'None
specified'}\n\n`;
});

context += "\nPEOPLE TO FOLLOW UP WITH:\n";
people.forEach((p, i) => {
  context += `${i+1}. ${p.properties?.Name?.title?.[0]?.plain_text || 'Unknown'}\n`;
  context += `   Follow-up: ${p.properties?.['Follow-ups']?.rich_text?.[0]?.plain_text || 'None'}\n\n`;
});

context += "\nTASKS DUE:\n";
admin.forEach((a, i) => {
  context += `${i+1}. ${a.properties?.Name?.title?.[0]?.plain_text || 'Untitled'}\n`;
  context += `   Due: ${a.properties?.['Due Date']?.date?.start || 'No date'}\n\n`;
});

return { context: context };
```

Step 6.7: Generate the Digest with AI

The Daily Digest Prompt
You are a personal productivity assistant. Generate a concise daily digest based on the following data.

{{context}}

TODAY'S DATE: {{current_date}}

INSTRUCTIONS:
Create a digest with EXACTLY this format. Keep it under 150 words total.

🌞 **Good morning!**

** 🎯 Top 3 Actions Today:**
1. [Most important/urgent action from projects or admin]
2. [Second priority]
3. [Third priority]

** 👥 People to Connect With:**
- [Person name]: [Brief follow-up reminder]

** ⚠️ Watch Out For:**
[One thing that might be stuck, overdue, or getting neglected]

** 💪 One Small Win to Notice:**
[Something positive or progress made, or encouraging thought]

RULES:
- Be specific and actionable, not motivational
- Prioritize overdue items and concrete next actions
- If there's nothing in a section, omit it entirely
- Keep language direct and practical
- Don't add explanations or commentary outside the format

Step 6.8: Send the Digest via Slack DM

- App: Slack
- Event: Send Direct Message
- To: Your Slack user (select yourself)
- Message Text: `{{AI Response}}`

Part 7: Build the Weekly Review Automation

Step 7.1: Create a New Zap

Name it: "Second Brain - Weekly Review"

Step 7.2: Set Up the Trigger

App: Schedule by Zapier
Event: Every Week

- Day of Week: Sunday
- Time of Day: 4:00 PM (or whenever works for you)

Step 7.3: Query This Week's Inbox Log

- Database: Inbox Log
- Filter: Created is within the last 7 days
- Limit: 50

  Token Limits: If you capture more than ~50 items per week, you may hit token limits. Either reduce the limit, split into multiple queries, or use GPT-4o/Claude 3.5 Sonnet which have larger context windows.

Step 7.4: Query All Active Projects

- Database: Projects
- Filter: Status equals "active" OR "waiting" OR "blocked"
- Limit: 30

Step 7.5: Format the Data

```
const inboxLog = JSON.parse(inputData.inbox_log || '[]');
const projects = JSON.parse(inputData.projects || '[]');

let context = "=== ITEMS CAPTURED THIS WEEK ===\n\n";

inboxLog.forEach((item, i) => {
  const originalText = item.properties?.['Original Text']?.title?.[0]?.plain_text || 'No text';
  const filedTo = item.properties?.['Filed To']?.select?.name || 'Unknown';
  const destName = item.properties?.['Destination Name']?.rich_text?.[0]?.plain_text || '';
  const status = item.properties?.Status?.select?.name || '';

  context += `${i+1}. [${filedTo}] ${destName || originalText.substring(0, 50)}\n`;
  if (status === 'Needs Review') {
```

```
    context += `    ⚠️ NEEDS REVIEW\n`;
  }
  context += `\n`;
});

context += "\n=== ACTIVE PROJECTS STATUS ===\n\n";

projects.forEach((p, i) => {
  const name = p.properties?.Name?.title?.[0]?.plain_text || 'Untitled';
  const status = p.properties?.Status?.select?.name || 'Unknown';
  const nextAction = p.properties?.['Next Action']?.rich_text?.[0]?.plain_text || 'None specified';

  context += `${i+1}. ${name}\n`;
  context += `   Status: ${status}\n`;
  context += `   Next: ${nextAction}\n\n`;
});

// Count by category
const categoryCounts = {};
inboxLog.forEach(item => {
  const cat = item.properties?.['Filed To']?.select?.name || 'Unknown';
  categoryCounts[cat] = (categoryCounts[cat] || 0) + 1;
});

context += "\n=== CAPTURE SUMMARY ===\n";
for (const [cat, count] of Object.entries(categoryCounts)) {
  context += `${cat}: ${count}\n`;
}

return { context: context, total_captures: inboxLog.length };
```

Step 7.6: Generate the Weekly Review

The Weekly Review Prompt

You are a personal productivity assistant conducting a weekly review. Analyze the following data and generate an insightful summary.

{{context}}

TOTAL CAPTURES THIS WEEK: {{total_captures}}

INSTRUCTIONS:
Create a weekly review with EXACTLY this format. Keep it under 250 words total.

🗓️ **Week in Review**

📊 Quick Stats:
- Items captured: [number]
- Breakdown: [x people, y projects, z ideas, w admin]

🎯 What Moved Forward:
- [Project or area that made progress]
- [Another win or completion]

🔴 Open Loops (needs attention):
1. [Something blocked, stalled, or waiting too long]
2. [Another concern]

💡 Patterns I Notice:
[One observation about themes, recurring topics, or where energy is going]

📌 Suggested Focus for Next Week:
1. [Specific action for highest priority item]
2. [Second priority]
3. [Third priority]

🔧 Items Needing Review:
[List any items still marked "Needs Review" or flag if none]

RULES:
- Be analytical, not motivational
- Call out projects that haven't had action in over a week
- Note if capture volume was unusually high or low
- Suggest concrete next actions, not vague intentions
- If something looks stuck, say so directly
- Keep language concise and actionable

Step 7.7: Send the Review via Slack DM

- To: Yourself
- Message Text: `{{AI Response}}`

Troubleshooting & Error Handling

Common Issues and Fixes

Issue	Solution
Slack connection expired	Go to My Apps → Slack → Reconnect. Happens every few months.
Notion says "Page not found"	Reconnect Notion and re-share your Second Brain page. Double-check database IDs.
AI returns invalid JSON	The Code step should handle markdown wrapping. If still failing, consider switching from GPT-4o-mini to GPT-4o.
AI classification consistently wrong	Add few-shot examples to your prompt.
Daily digest is empty	Check Notion queries are finding items. Verify filter conditions are spelled correctly.
Zap runs but nothing in Notion	Check Zap history for errors. Verify field names match exactly (case-sensitive).
Fix button not working	Verify Fix Zap triggers on threaded replies. Check thread_ts is captured correctly.

Monitoring Your System

Weekly health check (2 minutes):

1. Open your Inbox Log in Notion
2. Filter by Status = "Needs Review"
3. Process any stuck items
4. Filter by Created = Last 7 days
5. Spot-check that filings look correct

Monthly maintenance:

1. Check Zapier task usage (don't exceed your plan)
2. Review Zap run history for recurring errors
3. Archive completed projects to keep queries fast
4. Update prompts if classification quality drifts

Maintenance & Restart Guide

When You Fall Off (and you will)

Life happens. You'll have a week where you don't capture anything. Here's how to restart without guilt:

The 10-Minute Brain Dump:

1. Open your `sb-inbox` channel
2. Set a 10-minute timer
3. Dump everything in your head, one message per thought
4. Don't worry about being organized — just capture
5. The system will process overnight (or immediately)
6. Resume normal capturing tomorrow

What NOT to do:

- Don't try to "catch up" on weeks of missed captures
- Don't reorganize your Notion databases
- Don't feel guilty about the gap
- Don't add complexity to "prevent this from happening again"

Quarterly Cleanup (30 minutes)

1. **Archive completed projects:** Filter by Status = "Done" and move to Archive
2. **Review stale items:** Filter by Last Touched older than 30 days
3. **Clean up Inbox Log:** Archive entries 3+ months old
4. **Update People database:** Clear completed follow-ups

Upgrading the System

Rule: Only add complexity when you feel genuine pain, not when you feel clever.

Safe additions after 1 month:

- Voice capture (Slack mobile app voice messages)
- Email forwarding to Slack

- Tags vocabulary (add common tags to your prompt)

Safe additions after 3 months:

- Meeting prep automation
- Birthday/anniversary reminders
- Weekly email summary

Probably never needed:

- More than 4 destination databases
- Complex tagging taxonomies
- Custom AI models
- Multiple capture channels

Quick Reference Card

Daily Use

- **Capture:** Open Slack → `#sb-inbox` → Type thought → Send
- **Fix:** Reply in thread → `fix: this should be people`

What Arrives Automatically

- **Every capture:** Slack reply confirming what was filed
- **Every morning:** Daily digest with top 3 actions
- **Every Sunday:** Weekly review with patterns and priorities

Database Fields Cheat Sheet

Database	Key Fields
People	Name, Context, Follow-ups, Last Touched
Projects	Name, Status, Next Action, Notes
Ideas	Name, One-Liner, Notes
Admin	Name, Due Date, Status

Inbox Log Original Text, Filed To, Confidence, Status

Build time: 1-2 hours
Maintenance: 5 minutes/week
Return: Every thought captured, classified, and surfaced without cognitive overhead