WOMEN DEFINING AI · THINKING IN RAG

RAG Build Checklist

Everything you need to go from an idea to a working RAG system. Do the stages in order. Stages 1 to 5 get you a working build. Stage 6 is for when people will act on the answers. Stages 7 and 8 are for sharing it with other people.

STAGE 0 · BEFORE YOU BUILD

Decide what you're building

Half an hour of thinking here saves days of rebuilding later.

- [ ] **Problem statement written**
  "[Who] keeps asking [what], and the answer lives in [where]."
- [ ] **20 to 30 test questions with expected answers**
  Write the answer you expect and where it lives. Mix easy (one place), hard (two or more documents) and trap (not in your data) questions.
- [ ] **Tried it in a Claude or ChatGPT Project first**
  If a Project answers your test questions well, you may not need to build anything.
- [ ] **Level picked**
  Level 1 (keyword search), Level 2 (search by meaning, with embeddings) or Level 3 (add-ons like hybrid search and re-ranking). Start lower than you think.

STAGE 1 · ACCOUNTS AND KEYS

What you need to build

A Project bundles all of this for you. When you build, you assemble it yourself.

- [ ] **LLM API account and key**
  Anthropic (Claude) or OpenAI, or a local model through Ollama if data can't leave your machine. A chat subscription is not the same as API access; the API is billed separately per token.
- [ ] **A dedicated embedding model (not a chat model)**
  Needed for Level 2 and up. It turns text into numbers for search; Claude and GPT can't do this step. Options: OpenAI `text-embedding-3-small`, Voyage AI, Cohere, or local `nomic-embed-text` through Ollama. Anthropic doesn't make its own, so pair Claude with one of these. Use the same embedding model for storing and searching; switching means re-embedding everything.
- [ ] **Spending limit set on every API account**
  Set a monthly budget cap before you write any code.
- [ ] **Keys stored safely**
  In an environment variable or `.env` file that's listed in `.gitignore`. Never paste a key into code you share or push to GitHub.

- [ ] **A place to run code**
  Google Colab or Replit in your browser, or your laptop with an AI coding tool (like Claude Code, Cursor or Codex). Python has the most RAG examples.

STAGE 2 · YOUR DATA

Get the data ready

> Check first: are you allowed to send this data to a third-party API? If not, plan for local models and embeddings.

- [ ] **Access and permission confirmed**
  You can export it, and the data owner is fine with how you'll use it.
- [ ] **Collected in one place**
  One folder. Note which PDFs are scanned images; those need OCR before any text comes out.
- [ ] **Old versions and duplicates removed**
  Or tagged with a date so you can filter them.
- [ ] **Converted to Markdown where possible**
  Markdown keeps headings, lists and tables as plain text, which makes chunking and retrieval cleaner. Tools like MarkItDown or Docling convert PDFs and Word files; spot-check the output.
- [ ] **Structure cleaned up**
  Real headings, numbered steps, one term per thing, acronyms spelled out once, a caption line for each image, tables kept whole.
- [ ] **File names carry meaning**
  `Pump_Troubleshooting_Guide.pdf`, not `doc_final_v3.pdf`. Names become metadata.

STAGE 3 · INGESTION PIPELINE

Turn documents into searchable chunks

- [ ] **Loader extracts clean text**
  Open a few outputs and read them. Garbled tables and repeated headers show up here.
- [ ] **Chunking set**
  Start at 300 to 800 tokens with 10 to 20% overlap. Split on headings first, then paragraphs.
- [ ] **Metadata on every chunk**
  Document title, section, page and date, so answers can cite sources.
- [ ] **Chunks embedded and stored**
  NumPy or FAISS for small sets, Postgres with pgvector if you already use Postgres, or a vector database (Chroma, Qdrant, Pinecone).
- [ ] **Update plan written down**
  How a new or changed document gets re-chunked and re-embedded, and how a deleted one gets removed.

STAGE 4 · RETRIEVAL AND ANSWERS

Search, then answer

- [ ] **Search returns the top 3 to 5 chunks**
  Print them. If the right chunk isn't there, no prompt will fix the answer.
- [ ] **Prompt grounds the model**
  Retrieved chunks wrapped in tags with their source. Instructions to answer only from them, cite the source, and say "I couldn't find that" when the answer isn't there.
- [ ] **Conflicts handled**
  Tell the model to show both answers and say which source is newer.
- [ ] **Sources shown to the user**
  So people can check the answer themselves.

STAGE 5 · TESTING

Prove it works

- [ ] **Full test set run and scored**
  For each question: found it (right source in the top results), grounded, correct, honest on trap questions. A spreadsheet is enough: download the test sheet at madinagbotoe.com/ai-lab/rag-guide.
- [ ] **One change at a time, then rerun**
  Chunk size, top-k, prompt wording. Change one thing so you know what helped.
- [ ] **Upgrades only where tests fail**
  Missing exact codes: add hybrid search. Right chunk found but ranked low: add re-ranking. Answers missing context: try parent-child chunks.

STAGE 6 · VALIDATION

Prove it, and keep proving it

Testing shows it works today. Validation shows it's reliable enough to trust and stays that way. Do the tier that matches your stakes.

- [ ] **Tier 1 · Test set saved as a regression suite**
  Rerun it after every change to chunking, prompt, model or data, and compare scores to the last run.
- [ ] **Tier 1 · Automated scoring**
  An AI judge grades the answers: Copilot Studio's Agent Evaluation if you use Copilot, or RAGAS and TruLens for your own build. Hand-grade a sample to check the judge agrees with you.
- [ ] **Tier 1 · Model version pinned**
  Use a specific model version, not a "latest" alias, and rerun the suite before switching.

- [ ] **Tier 2 · Pass bar written before testing**
  For example: every answer cites a source, all trap questions get "not found," a set percentage of answers rated correct by an expert.
- [ ] **Tier 2 · Expert review and sign-off**
  Someone who knows the subject reviews a sample of answers and signs off before launch.
- [ ] **Tier 2 · Ongoing sampling**
  Review a sample of live answers on a schedule, plus every thumbs-down.
- [ ] **Tier 3 · Intended use and risk documented**
  What it's for, what it's not for, and what happens if an answer is wrong. Needed for healthcare, finance, HR and legal uses.
- [ ] **Tier 3 · Requirements traced to tests**
  Each requirement links to the test that proves it, with results kept as evidence.
- [ ] **Tier 3 · Change control and audit trail**
  Revalidate when the model, prompt or data changes, and keep logs. Check your industry's rules and the NIST AI Risk Management Framework.

STAGE 7 · PUTTING IT ON A SITE

Only if other people will use it

> Never put an API key in browser code. Anyone can open dev tools, copy it and spend your money.

- [ ] **A backend holds the keys**
  A small server or serverless function (Vercel, Netlify, Cloudflare Workers, or a Python FastAPI app) that the website calls. The backend talks to the LLM and the database.
- [ ] **Vector store hosted somewhere persistent**
  Serverless functions don't keep files between requests, so an in-memory FAISS index won't survive. Use a hosted option such as Supabase (pgvector) or Pinecone.
- [ ] **Chat interface built**
  Question box, answer, and the sources it used.
- [ ] **Login and access control, if the docs are private**
  Filter chunks by who's allowed to see them before they reach the model.
- [ ] **Users told what happens to their questions**
  A short note on what's logged and which AI provider sees it.

STAGE 8 · BEFORE REAL USERS

Keep it safe and affordable

Guide Part 10. Add these before you share the link widely.

- [ ] **Rate limits**
  For example, 10 questions per minute per user, so one person can't run up your bill.
- [ ] **Cost tracking and caps**
  Log tokens per request. Set daily and monthly limits in the app as well as on the API account.
- [ ] **Input checks**
  Catch prompt injection ("ignore previous instructions"), and strip personal data like emails and phone numbers before logging.
- [ ] **Fallbacks and timeouts**
  If the API is down or slow, show a friendly message or try a backup model.
- [ ] **"Not enough context" check**
  If search finds nothing relevant, say so instead of letting the model guess.
- [ ] **Caching**
  Reuse answers to repeat questions to save time and money. Clear the cache when you re-index.
- [ ] **Logging and feedback**
  Log questions, retrieved chunks and answers. Add thumbs up and down. Review unanswered questions to find gaps in your data.

OFFICE HOURS

Stuck? Bring this

1. Your problem statement and the path you plan to start with
2. 3 to 5 sample files
3. Your test questions, expected answers and where they live
4. What broke, and which "common issues" row it matches
5. Anything stopping you from starting this week