**Final Stack (fixed)**

* **Runtime & package mgmt:** Python 3.11 + uv (fast, deterministic), ruff (lint/format).
* **Model host:** **Ollama** (Windows native or WSL2).
* **Chat LLM:** **aya-expanse:8b via Ollama**. Pull once:  
  ollama pull aya-expanse:8b([Ollama](https://ollama.com/library/llama3.1/tags?utm_source=chatgpt.com))
* **Tool calling:** LangChain ChatOllama with .bind\_tools(...) (OpenAI-functions style). ([LangChain](https://python.langchain.com/docs/how_to/tool_calling/?utm_source=chatgpt.com), [js.langchain.com](https://js.langchain.com/docs/integrations/chat/ollama_functions/?utm_source=chatgpt.com))
* **RAG embeddings:** **BAAI/bge-m3** (multilingual, strong), via sentence-transformers. (Local, GPU if available, CPU fallback.)
* **Vector store:** **Chroma** (local, simple, stable).
* **Reranking (quality boost):** **BAAI/bge-reranker-base** (CrossEncoder) via sentence-transformers.
* **Document loaders:** PyPDFLoader, TextLoader, Docx2txtLoader from LangChain.
* **Backend:** **FastAPI** + SSE (server-sent events) for streaming; LangChain Runnable graph for orchestration.
* **Agent tools (3 you’ll demo):**
  1. **RAG** over user docs (primary)
  2. **Web search** via **Tavily** (free API key) for freshness
  3. **Python mini-sandbox** (math/data helper; subprocess + timeouts, restricted env)
* **Observability:** **Langfuse** (free tier) for traces/spans.
* **Evaluation:** **RAGAS** for answer faithfulness/context precision; small synthetic eval set. ([docs.ragas.io](https://docs.ragas.io/en/latest/getstarted/evals/?utm_source=chatgpt.com), [GitHub](https://github.com/explodinggradients/ragas?utm_source=chatgpt.com), [Langfuse](https://langfuse.com/guides/cookbook/evaluation_of_rag_with_ragas?utm_source=chatgpt.com))
* **Frontend:** **React + Vite + TypeScript + Tailwind + shadcn/ui**; token-streaming, citations, collapsible “tool calls,” drag-and-drop documents, and a side panel for traces.

**Repo layout (fixed)**

/app

/backend

main.py # FastAPI + SSE

settings.py

agent\_graph.py # LangChain Runnable graph

tools/

rag.py # retriever, reranker

web\_search.py # Tavily wrapper

py\_eval.py # sandboxed Python tool

rag/

ingest.py # loaders + chunk + embed + upsert

retriever.py

rerank.py

eval/

build\_evalset.py

run\_ragas.py

logs/

/frontend

index.html

src/

main.tsx

App.tsx

components/

Chat.tsx

Message.tsx

Sources.tsx

ToolCalls.tsx

Upload.tsx

Traces.tsx

lib/sse.ts

/ops

start.ps1 / start.sh # one-liner start

README.md

**17-Day Plan (deliverables every day)**

**Day 1 — Environment & model**

* **Install:** Python 3.11, uv, ruff, Node 20+, Ollama (Windows native).
* **Pull model:** ollama pull aya-expanse:8b . Verify chat locally with ollama run. ([Ollama](https://ollama.com/library/llama3.1/tags?utm_source=chatgpt.com))
* **Scaffold repos:** create /app/backend (FastAPI) and /app/frontend (Vite React).
* **Deliverable:** Hello-World FastAPI endpoint + Vite app served, both running.

**Day 2 — LangChain + Ollama chat (streaming)**

* **Add:** langchain, langchain-ollama, sse-starlette.
* **Implement:** /chat/stream SSE endpoint calling ChatOllama and streaming tokens to UI.
* **Frontend:** streaming chat box (auto-scroll, markdown render).
* **Deliverable:** Smooth token streaming in UI from **Llama 3.1**.

**Day 3 — RAG embeddings & DB**

* **Install:** sentence-transformers, chromadb
* **Choose embedder:** **BAAI/bge-m3**; create embedding service (GPU if available).
* **Chunking:** RecursiveCharacterTextSplitter (e.g., 700 overlap 100).
* **Deliverable:** /rag/ingest script that indexes PDFs/TXT/DOCX into **Chroma**; persisted DB folder.

**Day 4 — RAG retriever + citations**

* **Retriever:** Chroma.as\_retriever(k=5).
* **Augment:** Build prompt with selected chunks; show **source cards** in UI.
* **Deliverable:** Ask a question about your docs → answers with inline citations.

**Day 5 — Reranker (quality pop)**

* **Install:** sentence-transformers CrossEncoder **BAAI/bge-reranker-base**.
* **Pipe:** retriever k=10 → rerank to k=5 → pass to LLM.
* **Deliverable:** Side-by-side compare w/ without reranker; keep reranker **ON**.

**Day 6 — Agent tool #1: RAG as a tool**

* **Design:** Tool schema “retrieve\_knowledge(query) → passages”.
* **Bind tools:** ChatOllama().bind\_tools([...]) so the model decides when to call retrieval. ([LangChain](https://python.langchain.com/docs/how_to/tool_calling/?utm_source=chatgpt.com))
* **UI:** Collapsible “Tool calls” panel showing function name + args + result.
* **Deliverable:** Model autonomously calls RAG tool when needed.

**Day 7 — Agent tool #2: Web search (freshness)**

* **Setup:** Tavily free key; LangChain TavilySearchResults.
* **Bind tool:** web\_search(query, k=5); add simple snippet→url view in UI.
* **Deliverable:** Questions beyond local docs trigger web search tool automatically.

**Day 8 — Agent tool #3: Python mini-sandbox**

* **Implement:** Subprocess runner with timeout=5s, whitelist math, statistics, pandas optional; strip file I/O & network.
* **Bind tool:** py\_eval(code:str) -> result:str.
* **Deliverable:** The model can do small calcs/DF summaries safely and show the result.

**Day 9 — Orchestration polish & guardrails**

* **Prompting:** System prompt with clear tool-use guidelines and safety rails.
* **Refusals:** Basic filters (e.g., deny file system write); sanitize tool args.
* **Deliverable:** Stable tool selection; graceful fallbacks; nice error toasts in UI.

**Day 10 — Observability (Langfuse) & logs**

* **Integrate:** Langfuse Python/JS SDKs for traces/spans of each message/tool call.
* **UI:** “Traces” side panel links to Langfuse trace IDs.
* **Deliverable:** You can open any chat and see the full tool chain timeline.

**Day 11 –** Conversational Memory and dynamic context window management.

**Day 12 — RAG evaluation (RAGAS)**

* **Build evalset:** Script to auto-generate ~25 Qs from your corpus (LLM-assisted) + store refs.
* **Run:** ragas metrics (faithfulness, context precision/recall, answer relevancy). ([docs.ragas.io](https://docs.ragas.io/en/latest/getstarted/evals/?utm_source=chatgpt.com))
* **Deliverable:** CSV report + charts (top failure cases identified).

**Day 13 –** **Basic user management**

Implementing basic user management functionality including registration and login, saving conversations and chat history including allocating conversation memory and a context window in each conversation.

**Day 14— Data & UX polish**

* **Corpus:** Curate 10–20 high-quality PDFs relevant to your department (AI/ML papers / your notes).
* **UI polish:** shadcn/ui cards, keyboard shortcuts, copy-to-clipboard, code blocks, light/dark.
* **Deliverable:** Professional-looking UI with branded title and clean spacing.

**Day 15 — “Wow” features & demo scripts**

* **Add:**
  + Drag-and-drop new docs → live re-ingest.
  + Toggle “show reasoning steps” (just tool call summaries, not chain-of-thought).
  + One-click “export conversation as PDF” with sources.
* **Deliverable:** A 3-minute guided demo script that hits RAG + web + python tool.

**Day 16 — Hardening & performance**

* **Cache:** Simple in-memory prompt+retrieval cache.
* **Preload:** Warm the embedder & reranker at startup.
* **Stress:** 10 fast queries in a row; ensure no crashes, steady token stream.

**Day 17 — Final report drafting (tech + research)**

* **Write:** Methods, architecture diagram, tools used, prompts, eval methodology (RAGAS), results, limitations, future work.
* **Include citations:** Llama 3.1 tag page, LangChain tool-calling docs, RAGAS docs. ([Ollama](https://ollama.com/library/llama3.1/tags?utm_source=chatgpt.com), [LangChain](https://python.langchain.com/docs/how_to/tool_calling/?utm_source=chatgpt.com), [docs.ragas.io](https://docs.ragas.io/en/latest/getstarted/evals/?utm_source=chatgpt.com))
* **Deliverable:** First complete draft (3000–5000 words) + figures exported from UI and Langfuse screenshots.

**Day 18 — Slide deck & full rehearsal**

* **Slides (10–12):** Problem → Design → Demo → Evals → What’s next.
* **Record:** 90-second no-internet fallback demo (screen capture) in case of Wi-Fi hiccups.
* **Deliverable:** Final slides + backup video; timed run-through ≤ 8 minutes.

**Day 19 — Code freeze, cleanup, and showtime kit**

* **Freeze:** Tag v1.0.
* **README:** one-command start scripts (start.ps1 / start.sh).
* **Checklist:** Pull model cached, Chroma persisted, Tavily key set, sample docs ingested, Langfuse credentials set, Zoom audio checks.
* **Deliverable:** “Showtime kit” folder with slides, report PDF, backup video, and demo script.

**Concrete package list (pin these)**

**Backend**

* fastapi, sse-starlette, uvicorn[standard]
* langchain, langchain-community, langchain-ollama
* sentence-transformers (for **bge-m3** + **bge-reranker-base**)
* chromadb
* pydantic, python-dotenv
* httpx, tenacity
* langfuse
* ragas, pandas, numpy

**Frontend**

* react, react-dom, vite, typescript
* tailwindcss, @radix-ui/react-icons, **shadcn/ui**
* react-markdown, remark-gfm, rehype-raw, prismjs (or shiki)
* axios (SSE via EventSource)

**“Impressive in Zoom” checklist**

* **Citations that jump on hover** (source snippet preview).
* **Collapsible tool calls** showing the function name + args + JSON return.
* **Traces side panel** (Langfuse) → click through to show spans.
* **RAGAS chart** (small bar chart of metrics) and explain improvements from reranker.
* **Drag-drop ingest** → ask a question → instant answer with new citations.
* **Export chat as PDF** for the head of department.

**Notes you can say out loud (with receipts)**

* “The model is **aya-expanse:8b** running locally in **Ollama**; you can see the exact tag on the official library page.” ([Ollama](https://ollama.com/library/llama3.1/tags?utm_source=chatgpt.com))
* “I implemented **function calling** with LangChain’s .bind\_tools() so the model decides when to retrieve, browse, or calculate.” ([LangChain](https://python.langchain.com/docs/how_to/tool_calling/?utm_source=chatgpt.com))
* “I evaluated retrieval quality with **RAGAS** (faithfulness, context precision/recall) and iterated with a reranker to boost precision.” ([docs.ragas.io](https://docs.ragas.io/en/latest/getstarted/evals/?utm_source=chatgpt.com))

**One-command dev start (target for Day 17)**

* **Windows PowerShell:** .\ops\start.ps1
  + Starts Ollama (if not running), FastAPI (backend), Vite (frontend).
* **WSL/Linux/macOS:** bash ./ops/start.sh

If you want, next step I can generate:

1. the exact requirements.txt (pinned versions),
2. FastAPI SSE endpoint + LangChain graph boilerplate, and
3. the Vite + Tailwind + shadcn scaffold with a ready-made streaming chat.