Lesson 6 · Repo-aware AI Assistant

PDF: this lesson · **Install (Linux · macOS · Windows):** guide

> **Part of local-ai-lab** - a hands-on course for building local AI.

> **Interactive version (slides):** https://nikolareljin.github.io/local-ai-lab/lesson-6-repo-aware-assistant.html **Read it locally (no GitHub Pages):** `./run -l 6 lesson` **Course home:** https://nikolareljin.github.io/local-ai-lab/ **Source:** https://github.com/nikolareljin/local-ai-lab **Author:** Nik Reljin **Time:** ~30-45 min · **Prerequisites:** Lesson 1 (Lessons 3-5 helpful) · full objectives in SYLLABUS.md

> **Lessons:** 1 · RAG → 2 · MCP → 3 · Hybrid retrieval → 4 · RAG safety → 5 · RAG evaluation → **6 · Repo-aware assistant (you are here)** → 7 · LangChain → ... → 15 · Docs from changes

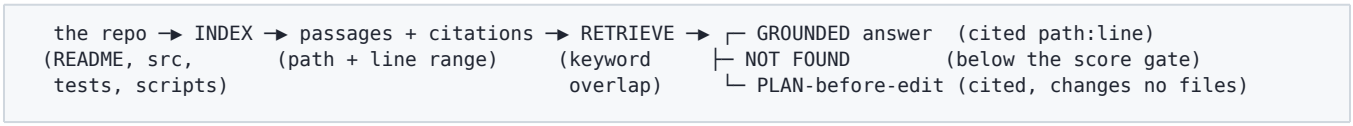
> **Status: working demo.** Runnable in **Python, Node.js, and C# / .NET** - same algorithm, three languages, identical output. Runs 100% offline, no model required. See **From demo to production** at the end for what to harden for real use.

What you'll learn

Lessons 1-5 built retrieval, made it better, safe, and measurable. This lesson points all of that at **one repository** and adds the two habits that make a code assistant trustworthy:

> **Cite every answer, and refuse when the answer isn't in the repo.**

An assistant that answers from your codebase is only useful if you can check it - and only safe if it knows when to say **I don't know**.



By the end you'll understand:

- **Indexing for citations** - splitting files into passages that each remember their `path:start-end`
- **Grounded answers** - answering **only** from retrieved lines, always cited
- **Abstention** - returning "not found" when nothing clears a minimum score, instead of guessing
- **Plan-before-edit** - turning a change request into relevant files → behaviour → change → tests → docs, without touching a single file

> **The one idea:** a repo-aware assistant is retrieval plus discipline. The retrieval you already have; the discipline is **cite or abstain**, and **plan before you edit**.

The demo

A tiny sample repo lives under `data/repo/` - a `notes-api` project with a README, three source files, a test, and a script. The questions that drive the demo are in `data/questions.json`: two the repo can answer, one change request, and one the repo knows nothing about.

The assistant **indexes** every file into line-numbered passages (each carrying its citation), **retrieves** the passages that overlap a question, and then either **answers** from them (cited), **abstains** ("not found"), or returns a **plan**.

Run it (offline, no dependencies)

From the repo root - the `demo` action prints the result and exits, with **nothing to install** (pure standard library). Pick any language; all three give the **same** output:

```
./run -l 6 demo                 # Python - index the repo, answer all four questions, exit
./run -l 6 --lang node demo     # Node.js - same output
./run -l 6 --lang csharp demo   # C# / .NET 8 - same output
./run -l 6 test                 # the offline Python test
./run -l 6 lesson               # read this lesson in a browser, served locally
./run -l 6 show                 # walk through this lesson's steps (code, data, prompts, commands)
```

Output:

```
Repo-aware assistant  -  indexed 6 files, 25 passages under data/repo

Q1  where is chunking implemented
   GROUNDED  -  answered only from indexed repository lines
   src/chunker.py:1-1
     """Chunking is implemented here: turn a note into passages."""
   sources: src/chunker.py:1-1

Q2  which tests cover the retriever
   GROUNDED  -  answered only from indexed repository lines
   tests/test_retriever.py:1-1
     """Tests that cover the retriever: ranking order and tie-breaking."""
   sources: tests/test_retriever.py:1-1 . README.md:8-12 . src/chunker.py:4-9

Q3  where should i add a new embedding provider   [plan-before-edit]
   PLAN  -  no files changed, approve before editing
   1. relevant files    src/providers.py:1-1 . src/providers.py:8-10 . src/providers.py:13-15
   2. current behaviour  src/providers.py:1-1  ->  """Embedding providers. Add a new embedding provider by registering it h
   3. minimal change     add the new code alongside src/providers.py, matching the pattern already there
   4. update tests       tests/test_retriever.py
   5. update docs        README.md

Q4  how do i configure kubernetes autoscaling
   NOT FOUND  -  best match scored 1 (< min 2), so the assistant abstains
     no citation, no invented answer
```

Q1 and Q2 answer from real lines and cite them. Q3 returns a plan and changes nothing. Q4 asks about Kubernetes - nothing in this repo clears the score gate, so the assistant **refuses** rather than inventing an answer. **That refusal, not the answers, is the point of the lesson.**

Experiment in the playground (needs Flask)

For a hands-on feel, bare `./run -l 6` opens an interactive **assistant** over the same repo. Type a question and read the cited answer plus the passages behind it, or flip on **plan mode** - then move the sliders and watch a grounded answer turn into **not found**.

```
./run -l 6           # opens http://127.0.0.1:<port> - ask the sample repo
```

Control	What moves
`min_score` → 3-4	a real answer flips to **not found**: the gate is stricter than the evidence
`top_k`	how many passages are considered (and cited as sources)
plan mode on	any question returns a plan-before-edit instead of an answer

> The playground is a small Flask app, so unlike the `demo` it needs one dependency. `./run` installs it into the project venv automatically on first use.

> **Why deterministic keyword retrieval (not embeddings + a model)?** To keep the lesson **offline, reproducible, and byte-identical across Python, Node.js and C#** - the same constraint every lesson in this course honours. The **contract** is what matters: passages carry citations, answers come only from retrieved lines, and the assistant abstains below a score. Swap in embeddings and a model and that contract is unchanged - see **From demo to production**.

Concept 1 · Index for citations

You can't cite what you didn't record. Indexing splits each file into **passages** (here, blocks separated by blank lines) and stores each passage's `path` and 1-based line range. That pair **is** the citation:

```
src/chunker.py:1-1       <- path : start-end
```

Because the range is captured at index time, every passage retrieval returns already knows exactly where it came from - the citation is a property of the index, not an afterthought.

Concept 2 · Answer only from retrieved lines

The answer is drawn from the top retrieved passage and returned **with its citation** - never free-form text. A grounded answer you can click through to the source beats a fluent paragraph you have to fact-check.

Concept 3 · Abstain when it isn't there

The single most important branch in the demo:

```
if not hits or hits[0][0] < min_score:
   return {"kind": "not_found", ...}      # no citation, no invented answer
```

The failure mode of a code assistant isn't a wrong line number - it's a confident answer about code that doesn't exist. Gating on a score and saying **not found** below it is what stops that.

Concept 4 · Plan before you edit

A change request gets a **plan**, not an edit: relevant files, current behaviour (cited), a minimal change, and the tests and docs to touch. Producing it changes **no files**. **Propose** and **apply** stay separate, so a human reads the plan before anything touches disk.

Polyglot by design

The index, the retriever, and the answer/plan logic are language-agnostic, so this lesson ships in **Python, Node.js, and C# / .NET** - each dependency-free, each reading the same `data/repo/` corpus and `data/questions.json`, each producing byte-identical output.

Port	Entry point	Run
Python	`python/repo_assistant.py`	`./run -l 6 demo` · `./run -l 6 test`
Node.js	`node/repo_assistant.mjs`	`./run -l 6 --lang node demo`
.NET 8	`dotnet/Program.cs`	`./run -l 6 --lang csharp demo`

> All three commands are declared once in `lesson.json` - the single source of truth the `./run` engine reads. `./run -l 6 show` renders this lesson's elements in order. The interactive playground (`./run -l 6`) is Python-only, by convention shared across the course.

Extend it to your own repository

The demo indexes a bundled sample so the output stays reproducible - but the same code runs against **any** repo. Every port honours a `REPO_PATH` environment variable and two subcommands, `ask` and `plan`:

```
# ask your own repo (skips .git, node_modules, hidden/dot dirs, build output, and non-text files)
REPO_PATH=/path/to/your/repo python lessons/06-repo-aware-assistant/python/repo_assistant.py ask "where is auth handled?"
REPO_PATH=/path/to/your/repo python lessons/06-repo-aware-assistant/python/repo_assistant.py plan "where should I add a cac

# or the ready-made wrapper - indexes the current directory
cd ~/work/my-service && /path/to/lessons/06-repo-aware-assistant/extend/repo-ask "where is the server started?"
```

The Node and C# ports take the same `REPO_PATH` and `ask`/`plan` subcommands, so the whole tool is polyglot. **EXTEND.md** walks through three ways to take this further:

1. a **standalone** `repo-ask` **CLI** (`extend/repo-ask`) a team can share,

2. a **repo-search MCP tool** you register as a Claude Code skill (built on the Lesson 2 server), and
3. the port as a **drop-in library** - reuse `build_index` / `retrieve` / `answer` / `plan` and swap in BM25, embeddings, or a model while keeping the **cite-or-abstain** contract.

Exercises

- **Add a question:** add a locate question to `data/questions.json` whose answer is in `README.md`, and confirm the citation points there.
- **Find the gate:** raise `min_score` until a correct answer starts returning **not found**. What's the lowest score any real answer relies on?
- **Break a citation:** delete a blank line in a corpus file so two passages merge, and watch the cited line range change. Citations track the index, not the prose.
- **Plan a real change:** point the indexer at a folder of your own and ask "where should I add X?" - does the plan's **relevant files** list match where you'd actually start?

From demo to production

- **Keep the citation contract, swap the pipeline** - index your real repo, use BM25 / embeddings / the Lesson 3 hybrid and a model for the answer and plan; the `path:start-end` grounding and the abstain gate are unchanged.
- **Index more signal** - symbols, imports, call graphs, git blame and PR history, not just lines.
- **Make the model cite** - require every claim to reference a `path:line` from the retrieved set, and reject answers that cite nothing. That's the **not-found** rule, enforced on the model.
- **Keep plan and apply separate** - a plan proposes; applying is a second, human-approved step. Wire repo-search to the Lesson 2 MCP server so a host (Claude Code, an IDE) can call it as a tool.
- **Evaluate it** - fold these questions into Lesson 5's golden set so a drop in citation accuracy, or a lost **not-found**, shows up as a failed check instead of a bad answer in production.

Next lesson

Lesson 7 · Rebuild RAG with LangChain → - rebuild the pipeline on a framework, watch where it grounds differently, and price the swap in components replaced, lines you still write, and packages you did not read.
