Module 1: The AI-Augmented Environment, Lesson 2: IDE Integration and Tooling

Hands-On: Configuring Your AI-Native Workspace

Overview

This demo proves four claims from the lecture in real time:

- Native IDE integration eliminates the context-switch tax that costs 20+ minutes of deep-work recovery every time you alt-tab
- Zero-Data-Retention is a toggle you own, not a policy buried in a PDF
- A focused index beats a bloated one — indexing node_modules is indexing noise
- Cross-file semantic reasoning is the real test — if that works, the rest of the course works

You will install Cursor, lock down privacy, control the index with a .cursorignore file, and then prove the setup works by asking a cross-file question without opening either file. The student lab extends this to add a local Ollama endpoint for offline work.

Before You Start

- Close the browser tab where your AI currently lives — the point of this hands-on is to kill that tab
- Clone a sample repo you have read access to (any Python or JS project with 5+ files will do)
- Have at least 10 minutes uninterrupted — initial indexing needs to finish before Step 4
- Optional: temporarily disable GitHub Copilot to avoid dueling autocompletes during the demo

---

> **Remember**
>
> Native integration equals deep context. A browser-based AI is blind to your project — it cannot see your files, your imports, or your architecture. We are fixing that in the next 15 minutes, and you will feel the difference on your very next task.

During Hands-On — Keep Visible if Possible

```
1. ENGINE     — install Cursor, verify inline autocomplete + sidebar chat
2. PRIVACY    — toggle Zero-Data-Retention ON before you touch real code
3. INDEX      — create .cursorignore, exclude noise, rebuild the index
```

```
4. VALIDATE   — ask a cross-file semantic question without opening either
file
```

Step 1 — Install Cursor and Verify the AI Engine

> **What to Learn**
>
> We are picking the Native Route from the three options in the lecture — Cursor. The migration cost is essentially zero because Cursor is built on VS Code, so your extensions, keybindings, and themes carry over. What you gain is native codebase indexing and the agentic composer, neither of which exists in the plugin-on-top-of-an-editor model.

Do this:

- Download Cursor from cursor.com and run the installer
- Launch Cursor; sign in with your account; let it import your VS Code settings when prompted
- Open your sample repo as a workspace (File → Open Folder)
- Open any source file and start typing a function signature — wait for the ghost-text completion to appear
- Press Cmd+L (Mac) or Ctrl+L (Windows/Linux) to open the sidebar chat — ask: "What does this file do?"

> **What you should get**
>
> Ghost-text autocomplete appears within about half a second after you stop typing — accept it with Tab. The sidebar chat opens on the right and answers with specific references to functions, classes, and line numbers in the file you have open, not a generic "this file contains Python code" reply. That specificity is how you know the native integration is active.

You should ask:

"Why didn't my VS Code extensions break when I switched editors?"

Answer: Cursor is a fork of VS Code, not a rewrite. It reads the same settings.json, the same keybindings, and pulls from the same extension marketplace. That is the whole reason the migration cost is near zero — and it is also why you should never treat Cursor as a separate tool to learn. Treat it as VS Code that finally has a real brain.

Step 2 — Enable Zero-Data-Retention (The Privacy Mandate)

> **Remember**
>
> Before a single line of proprietary code passes through the AI, you must know where that code is going. Public models log prompts. Training pipelines ingest logs. Your secrets become someone else's dataset. This is why ZDR is non-negotiable on any professional codebase — and it is one toggle, so there is no excuse to skip it.

Navigate to the privacy settings:

```
Mac:      Cmd+Shift+J         →  Cursor Settings
Windows:  Ctrl+Shift+J        →  Cursor Settings

Then:     General  →  Privacy Mode  →  Enabled
Confirm:  "Prompts and code are not stored or used for training"
```

Verify it is actually on:

- Look at the status bar at the bottom of the window — a shield or "Privacy" badge should now be visible
- Hover the badge — it should read "Privacy Mode: Enabled"
- If you are on an enterprise plan, confirm your admin has not locked this setting to OFF at the org level

> **What you should get**
>
> The privacy badge in the status bar, plus the confirmation text in Settings. If the badge is missing or greyed out, stop here — do not proceed to Step 3 until ZDR is confirmed active. This is the one step in this hands-on that is worth redoing until it is right.

Tip: ZDR is not a substitute for never pasting secrets. It prevents training and storage, but the prompt still travels to the model provider in transit. Never paste API keys, connection strings, or .env contents into any AI prompt — local or cloud. That is what .cursorignore is for, and that is what we configure next.

Step 3 — Create .cursorignore to Focus the Index

> **What to Learn**

> The IDE builds a semantic index — a vector embedding — of your whole repository so it can answer project-wide questions. That index is only as useful as what you let it see. Index your node_modules folder and you are burning cycles embedding 40,000 files of framework code you will never ask about. Worse, you dilute the AI's attention on your actual source. Narrow index = fast index = accurate answers.

At the root of your repo, create a file named .cursorignore with this content:

```
# --- Dependencies (never index, never useful) ---
node_modules/
venv/
.venv/
__pycache__/
vendor/

# --- Build artifacts ---
dist/
build/
target/
*.pyc
*.pyo
*.class

# --- Secrets — NEVER let these touch the index ---
.env
.env.local
.env.*.local
*.pem
*.key
secrets/
credentials/

# --- Binaries, media, data dumps ---
*.zip
*.tar.gz
*.jpg
*.png
*.pdf
*.sqlite
*.db

# --- Auto-generated / lockfiles (noise, not signal) ---
package-lock.json
poetry.lock
```

```
yarn.lock
*.min.js
*.min.css
```

Save, then force a re-index so the new rules actually apply:

- Open the command palette: Cmd/Ctrl+Shift+P
- Type and run: "Cursor: Resync Index" (or "Rebuild Codebase Index" depending on version)
- Watch the bottom-right corner for the indexing progress indicator — wait for it to finish

> **What you should get**
> A .cursorignore file committed to your repo root and a rebuilt index that finishes in seconds instead of minutes. If indexing still takes a long time after the rebuild, open the ignored-paths view (Settings → Features → Codebase Indexing) and confirm your exclusions are being honored. A slow index on a small repo is almost always a missing .cursorignore.

Step 4 — Validate the Index with a Cross-File Semantic Query

> **Remember**
> This is the real test. Anyone can install an editor. Anyone can flip a toggle. But the whole reason we chose native integration over a browser tab was project-wide reasoning — so we are going to prove it works. You are going to ask a question that requires the AI to look at multiple files you did not open, did not paste, and did not mention by name.

Open sidebar chat (Cmd/Ctrl+L) and paste this query exactly:

- Find every place in this repo where a function named
- authenticate_user (or login, or verify_token) is CALLED.
- For each call site, give me:
-
- 1. The file path and line number
- 2. The function that contains the call
- 3. Whether the caller wraps the call in a try/except
- or equivalent error handler
-
- At the end, tell me which call sites are missing
- error handling and rank them by risk (auth failures

- in public-facing routes are highest risk).

What you should get

A structured answer that lists files you never opened, with accurate line numbers, grouped by whether they handle errors. The AI should cite at least two separate files in its response. If your repo does not contain an auth function, substitute any function you know is called from more than one place — the pattern is the same. The key behavior to observe is that the AI references file paths you never typed.

Tip: If the AI responds by asking you to paste the relevant files, your index is not working. Go back to Step 3, rebuild the index, and wait for it to finish before retrying. A working native integration never asks you to paste code that already exists in the repo.

Wrap-Up — Your Workspace Is Locked In

Remember

Four steps. One workspace. Engine verified, privacy toggled, index focused, cross-file reasoning proven. That is what the lecture meant by "deep integration equals deep context" — and you just built it in 15 minutes. Every lesson after this assumes this setup is done and working.

Key takeaways:

- Native beats browser — the context-switch tax is real, and deep integration is the only fix
- Zero-Data-Retention is a toggle you own, not a policy you read and forget
- A narrow index is a fast index — .cursorignore is a security tool and a performance tool
- Cross-file semantic query is the acceptance test; if that works, everything downstream works
- If you ever find yourself pasting a file into the AI, your setup has regressed — re-run Step 3