WEEK 4 JOB LIFE

Go Full-Stack — Neon + Auth

Learning Objectives

By the end of this session, students should be able to:

- **Provision a production Postgres database and an auth system by prompt** — Neon MCP creates the DB, Drizzle defines the schema, and Neon Auth wires Google login, all through one conversation.
- **Recognise and execute the OAuth-consent manual moment** — the one step the AI cannot do for you — without panicking or skipping ahead.
- **Ship a full-stack feature end-to-end** (signed-in visitors leaving messages on your site) in a single 2-hour class, even though you've written zero lines of server code.

Core Topics

- What "going full-stack" actually means when the AI writes the stack: a database, an auth layer, server actions, and a UI — all at once.
- Drizzle ORM as a type-safe schema definition the AI can reason about.
- Neon Auth — why it exists and why it's the fastest path from "no login" to "signed-in users" for beginners.
- Env-var plumbing across three environments (local, preview, production) and why AI doing this is a superpower.
- The OAuth trust dance: why Google requires human consent and where the callback URL must match.

Tools / Stack

Tool	Role this week
Neon Postgres (via MCP)	Serverless Postgres; AI provisions via Neon MCP.
Neon Auth	Drop-in auth with Google provider; AI wires it.
Drizzle ORM	Type-safe schema definition AI reads and extends.
Vercel CLI	Syncs env vars between Neon ⟶ local ⟶ production.
Google Cloud Console	The one place a human opens the OAuth consent screen.

Your personal site — cumulative features through this week:

`Wk 1` · Dev toolkit installed — Cursor, Claude Code, Gemini CLI, MCPs, Typst skill

`Wk 2` · Personal website deployed to Vercel

`Wk 3` ·

AI chat widget (Gemini 2.5 Flash) emulating you

`Wk 4` · **Neon Postgres + Neon Auth + Drizzle guestbook** ← NEW

Session Plan

Time	Activity
0 – 15 min	**Recap & Check-in.** Anyone's chat widget break this week? Quick triage. Also: who's had a stranger use their AI-clone yet?
15 – 40 min	**Concept Teaching.** What Postgres **is**, at the level a beginner needs. What an ORM does and doesn't do. Auth providers and callback URLs. Why env-var plumbing is a 20-year-old engineering pain we can now skip.
40 – 75 min	**Live Demo.** Instructor adds a full guestbook to her own site — Neon provisioning, Drizzle schema, Google login, protected UI — in 18 minutes of prompting. Class watches the whole flow.
75 – 105 min	**Hands-On Lab.** Students replicate on their site. The OAuth step is the only true human moment — rehearse it twice so nobody panics.
105 – 120 min	**Q&A + Wrap.** Common failure: callback URL mismatch. We'll debug one live.

Hands-On Lab

Task. By the end of class your site has a guestbook. Visitors click **Sign in with Google**, write a message, and see it appear (with their name + avatar) instantly on your About page. Refresh — the messages persist. This is a full-stack feature: database, auth, server action, UI.

PROMPT **Step 1 · Say to Cursor:**

> *Please provision a new Neon Postgres database for this project. Use the Neon MCP server. Name the project `my-portfolio-db`. When it's created:*
>
> 1. *Save the pooled connection string as `DATABASE_URL` in my `.env.local`.*
> 2. *Add the same `DATABASE_URL` to my Vercel production **and** preview environments via the Vercel CLI.*
> 3. *Install Drizzle ORM (`drizzle-orm`, `drizzle-kit`, `pg`) in my Next.js project.*
> 4. *Create `drizzle.config.ts` and a `db/schema.ts` file, ready for me to add tables.*
>
> *Do not create any tables yet — I want to do that in the next step. Report back with the database's region and the list of env vars you set.*

VERIFY **Step 2 · Verify:**

- Cursor confirms the Neon project is created and prints its region.
- Your `.env.local` has a `DATABASE_URL=postgresql://...` line.
- Running `vercel env ls` (Cursor will do this) shows `DATABASE_URL` for both `production` and `preview`.
- `drizzle.config.ts` and `db/schema.ts` exist.

PROMPT **Step 3 · Say to Cursor:**

> *Now define the guestbook schema in Drizzle. I want two tables:*
>
> - `users` *— a minimal profile table that Neon Auth can own. Columns:* `id` *(primary key, text),* `name` *(text),* `email` *(text, unique),* `image` *(text, nullable),* `created_at` *(timestamp default now).*
> - `guestbook_messages` *— columns:* `id` *(primary key, serial),* `user_id` *(text, FK →* `users.id` *, cascade on delete),* `body` *(text, not null, 1–500 chars),* `created_at` *(timestamp default now).*
>
> *Generate the migration SQL with* `drizzle-kit generate` *, show it to me in chat, then run* `drizzle-kit push` *to apply it against the Neon database. Confirm both tables exist by running a* `SELECT 1` *equivalent against each.*

VERIFY **Step 4 · Verify:**

- Cursor shows you the migration SQL before running it.
- The push completes without error.
- A verification query against both tables returns zero rows.

PROMPT **Step 5 · Say to Cursor:**

> *Install* ***Neon Auth*** *in this project. Configure it to use Google as the only sign-in provider, and to write authenticated users into my* `users` *table (the one we just defined). Add a* `Sign in with Google` *button component I can drop anywhere. Don't fill in the Google credentials yet — you'll need me to give you those in a moment. Tell me exactly what values to fetch from Google Cloud Console.*

MANUAL **Step 6 · Manual (human only):**

Cursor will tell you to open Google Cloud Console. In it:

1. Create a new project called `my-portfolio-auth` (or use an existing one).
2. Enable the **Google+ API** / **People API** if prompted.
3. Go to **APIs & Services → OAuth consent screen**. Choose **External**, fill in app name = `My Portfolio` , user support email = your email, save.
4. Go to **Credentials → Create Credentials → OAuth client ID**. Application type = **Web application**. Add the **Authorised redirect URIs** Cursor just printed (there will be two — one for localhost, one for production).
5. Copy the resulting **Client ID** and **Client secret**.

Why manual: Google requires a human to accept OAuth terms and manually paste redirect URIs into its web console. No API, CLI, MCP, or skill can do this — by Google's design.

PROMPT **Step 7 · Say to Cursor:**

Here are the Google OAuth credentials:

• Client ID: ***[paste]*** *• Client secret:* ***[paste]***

Please:

1. *Add* `GOOGLE_CLIENT_ID` *and* `GOOGLE_CLIENT_SECRET` *to both* `.env.local` *and Vercel (production + preview) via the CLI.*
2. *Wire them into Neon Auth's Google provider config.*
3. *Also generate a strong* `AUTH_SECRET` *and set it in all three environments.*
4. *Restart the dev server and confirm the* ***Sign in with Google*** *button on* `http://localhost:3000` *now starts the Google flow end-to-end.*

VERIFY **Step 8 · Verify:**

- Clicking **Sign in with Google** on localhost takes you to Google's consent screen; signing in redirects back to your site, signed in.
- A row appears in the `users` table with your Google name + email.
- No `GOOGLE_CLIENT_SECRET` or `AUTH_SECRET` references show up in any committed file (`git status` confirms `.env.local` is ignored).

RECOVER **Step 9 · If stuck, say to AI:**

The sign-in redirect returns `Error 400: redirect_uri_mismatch` *. Can you tell me the exact redirect URI Neon Auth is requesting, compare it to what's registered in Google Cloud Console, and walk me through fixing the mismatch in Google's dashboard? I'll copy-paste anything you need.*

PROMPT **Step 10 · Say to Cursor:**

Now build the guestbook feature on my `/about` *page:*

- *If the user is* ***not*** *signed in, show a small* ***Sign in with Google to leave a message*** *button above the messages list.*
- *If the user* ***is*** *signed in, show a compact form: a textarea (max 500 chars, character counter visible), a* ***Post*** *button, and a tiny "posting as {user.name}" label.*
- *Below the form (or button), render all messages newest-first. Each message shows the poster's avatar, name, and a relative time ("3 minutes ago"). My own messages get a subtle accent border.*
- *Use a Next.js* ***server action*** *for posting. Validate length + trim whitespace on the server. Return an optimistic UI update so the message appears instantly.*
- *Revalidate the* `/about` *page on successful post so anyone else visiting sees the new message.*

Use the site's existing Tailwind tokens and accent colour. Commit in two commits: one for the server action, one for the UI.

VERIFY **Step 11 · Verify:**

- On localhost, when signed out, you see the sign-in button.
- After signing in, the form appears. Post **"Hello from the future"**.
- The message appears instantly above the form with your avatar and **"just now"**.
- Refresh. The message is still there.
- Open an incognito window, sign in with a different Google account, post again. Switch back to your original window, refresh — the other user's message is visible.

PROMPT **Step 12 · Say to Cursor:**

Deploy to production. Confirm all env vars are set on Vercel prod, push the commits, and wait for the deploy to finish. Then: open the production URL, sign in with Google, and post a message. Confirm the message appears on the live site. Also check that visiting the live site in a private window shows the message (proving the Neon connection works in prod).

VERIFY **Step 13 · Verify:**

- Production deploy succeeds.
- Live site shows the guestbook.
- The message you posted during step 11 locally is **not** on production (different data). A new one you post on prod appears immediately.
- Your Neon dashboard shows rows in both `users` and `guestbook_messages` tables.

WARNING · Don't commit secrets — ever

If at any point you see Cursor about to write `GOOGLE_CLIENT_SECRET = "..."` into a source file, stop it: **"Don't put the secret in a source file. Put it in `.env.local` and set it via Vercel CLI only."** This is a muscle to build — many real-world production leaks start with an IDE autocompleting a secret into the wrong file.

Weekly Assignment

Build / Implement.

- A working guestbook on your live site: sign in with Google, post a message, see it persist across refresh and across browsers.

Requirements.

- Neon Postgres is the source of truth. No in-memory arrays.
- Drizzle schema committed to the repo.
- Google OAuth callback URLs work in both `localhost` and production.
- At least three messages on the live guestbook by submission time (ask classmates to post so you have real data).
- Screenshot of your Neon dashboard showing the `users` and `guestbook_messages` tables with rows.

Submission. Live URL + Neon screenshot in Slack before Week 5.

Resources

Docs

• Neon — MCP server + CLI
• Neon Auth — Next.js integration
• Drizzle ORM — schema basics + migrations
• Vercel env vars — per-environment management

Videos

• Instructor demo: "Full-stack in one conversation"

Repos

• `drizzle-team/drizzle-orm` — examples
• `neondatabase/neon` — docs repo

Real-World Application

This is the hinge class of the term. Before Week 4 you were shipping static sites with LLM calls. After Week 4 you're shipping **actual products** — signed-in users persisting state to a real database. The stack you learned today (Next.js + Neon + Drizzle + an auth provider) is what most AI startups in 2026 are running in production. You can now claim **full-stack developer** on LinkedIn without a shred of imposter syndrome.

> **CAREER**
>
> Recruiter skimming your LinkedIn pauses on "guestbook with OAuth + Postgres + Drizzle, deployed to Vercel" and thinks: **this person can ship features.** That's exactly the signal you want.

Challenges & Tips

- **"`drizzle-kit push` asks me to confirm a destructive change."** Read the diff in Cursor's output. If it's shrinking a column or dropping a table, stop and ask Cursor **"Why does this migration want to delete X? Is there a safer path?"** In a classroom DB it's usually fine; the habit of reading migration diffs is what matters.
- **"OAuth works on localhost but breaks in production."** The callback URL in Google Cloud Console is environment-specific. You need **both** `localhost:3000/api/auth/callback/google` **and** `your-project.vercel.app/api/auth/callback/google` registered. Ask Cursor to list both and fix them in the Google dashboard.
- **"My session disappears on refresh."** The auth cookie isn't being set on the production domain. Ask Cursor **"Check my Neon Auth cookie configuration — why isn't the session persisting on the live site?"**
- **"I posted a message but it doesn't show up."** Server-action revalidation missed. Say **"Add `revalidatePath('/about')` to the server action after the insert succeeds."**
- **"My Neon free tier is at 80%."** You're storing emoji-heavy messages or forgot to clean up a test table. Ask Cursor **"Show me my Neon storage usage and help me drop any unused test tables."**

> **STUCK? RESCUE**
>
> If auth fails in a way you don't understand, paste the **exact** error from the browser URL bar into Cursor. 95% of OAuth errors are encoded in that query string — Cursor can decode and fix.

Instructor Notes (Internal)

- **This class runs long for 30% of students.** Budget for the OAuth step to take 15 minutes for the slowest. Make the rest move on to step 10 in parallel if they're waiting.
- **The OAuth consent screen is the single most common stuck point of the term.** Have a pre-recorded 90-second clip showing the exact clicks in Google Cloud Console. Play it before step 6.
- **Walk the room during step 5.** Watch for students who skip straight to step 7 without doing step 6. Those are the students whose auth will mysteriously fail in step 8 and take 20 minutes to debug.
- **After class, post a "common auth errors" thread in Slack.** `redirect_uri_mismatch`, `invalid_grant`, cookie-SameSite issues — these will come up over the next few days as students visit the prod site from new networks.
- **Neon free tier limits.** Warn students: don't spin up multiple Neon projects. One per student, reused week after week.

Student Feedback

Challenges Observed

Extra Information

MEET YOUR INSTRUCTOR

Chan Meng

Full-Stack Engineer · AI Agent Builder · Minimalist

> *"Subtraction for life, addition for thought."*

AI AGENT BUILDER

AGENTIC SYSTEMS · LLM INTEGRATION

FEMTECH INNOVATOR

MINIMALIST CODE · UN CSW 69 SPEAKER

Chan builds **production-grade AI agents** at the intersection of **agentic systems, LLM integration, and women's health**. Between China and New Zealand, she ships systems that go to work for real users at real companies — and she believes code, like life, becomes most powerful when you subtract what isn't essential. A **Master of Applied Computing with Distinction** from Lincoln University, NZ (Dean's List), she was featured at **UN Women CSW69** for her work on AI and gender equality.

Currently · Founding Engineer at Gavigo · Senior Full-Stack Engineer at She Sharp (NZ's leading women-in-STEM nonprofit, 2,200+ members) · CTO of FemTech Weekend · Open Source Contributor to CopilotKit (24.6k★). For the full project portfolio and code, visit github.com/ChanMeng666.

Stay in the loop

NEWSLETTER · RECOMMENDED

"What's Shipping in AI" — a weekly curated digest, every Monday. chanmeng.org/#newsletter

LINKEDIN
chanmeng666

GITHUB
ChanMeng666

PORTFOLIO
chanmeng.org

EMAIL
chanmeng.dev@gmail.com

Building tools that matter. Let's connect.