WEEK 7 JOB COOL

Real-Time Slack Notifications

Learning Objectives

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

- **Create a Slack App and obtain an incoming webhook URL** — the minimum human work — then hand everything else to Cursor.
- **Build a contact form that writes to Neon and fires a Slack webhook in one server action**, through a single prompt describing the feature end-to-end.
- **Think about notification-as-observability**: why a tiny Slack ping on every real event is how senior engineers **actually** monitor small products in production.

Core Topics

- Slack Apps, Incoming Webhooks, and why this is the cheapest real-time pipe in the world.
- Server actions as the right "trust boundary" for webhook calls — the webhook URL is a secret.
- Fan-out pattern: one event, two destinations (database + Slack). What happens when one of them fails and how to decide whether to retry, log, or drop.
- Spam defence: rate limiting, honeypot fields, content filtering — protecting your Slack from getting pinged 400 times by a bored bot.

Tools / Stack

Tool	Role this week
Slack	Where notifications land — your own workspace.
Slack Incoming Webhooks	The simplest HTTP-POST-to-Slack mechanism.
Next.js server actions	Where the form handler lives; server-only secrets.
Drizzle + Neon	New `contact_submissions` table persists every form.
`@upstash/ratelimit` (optional)	Sliding-window rate limit for the form.

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

Wk 5 · Vercel Blob image uploads in guestbook

Wk 6 · MDX blog system with RSS

Wk 7 · **Real-time Slack notifications on contact form** ← NEW

Session Plan

Time	Activity
0 – 15 min	**Recap & Check-in.** Who got their first blog subscriber? Any surprising replies?
15 – 40 min	**Concept Teaching.** How Slack incoming webhooks work at the HTTP level (show Cursor `curl`-ing the webhook). Why the webhook URL is a secret. Fan-out + failure modes. Why your first "production monitoring" should be a Slack channel.
40 – 75 min	**Live Demo.** Instructor adds a contact form to her site and watches a classmate submit it — a Slack message pops up on the projector in real time. Class claps.
75 – 105 min	**Hands-On Lab.** Students build their own contact form; everyone submits **one** message to **one** classmate's form to test the real-time pings.
105 – 120 min	**Q&A + Wrap.** Anyone whose Slack ping didn't arrive gets debugged — usually a bad URL or missing env var.

Hands-On Lab

Task. By the end of class your live site has a **Contact** page with a form. When someone submits it, the message is saved to Neon **and** a real-time Slack message arrives in a `#my-site` channel in your own Slack workspace — within two seconds, with the sender's name, email, and message.

MANUAL **Step 1 · Manual (human only):**

Open api.slack.com/apps in a browser and:

1. Click **Create New App → From scratch**. Name it `My Portfolio Contact`. Select the workspace you want notifications to land in (your personal one is fine — create a new workspace if you don't have one).
2. In the app settings, click **Incoming Webhooks** → toggle **Activate Incoming Webhooks** on.
3. Click **Add New Webhook to Workspace**. Choose (or create) a channel called `#my-site-contact`. Approve.
4. Copy the resulting webhook URL (it looks like `https://hooks.slack.com/services/T.../B.../...`). Keep this tab open.

Why manual: Slack's app-creation and workspace-installation flows require human consent at several steps. Slack explicitly does not expose an API that can create apps on a user's behalf — by design.

PROMPT **Step 2 · Say to Cursor:**

Here is my Slack incoming webhook URL: ***[paste]****.*

Please:

1. *Add* `SLACK_CONTACT_WEBHOOK_URL` *to* `.env.local` *and to Vercel's production and preview environments via the Vercel CLI.*
2. *Install* `zod` *for input validation and* `@upstash/ratelimit` + `@upstash/redis` *for rate limiting (if I don't have Upstash set up yet, skip rate limiting for now — we'll add a simpler approach).*
3. *Add a new Drizzle table* `contact_submissions` *with columns:* `id` *(serial PK),* `name` *(text, 1–80 chars),* `email` *(text, must look like an email, validated with zod),* `message` *(text, 10–2000 chars),* `user_agent` *(text, nullable),* `ip_hash` *(text, nullable — we'll hash the IP, never store raw),* `created_at` *(timestamp default now).*
4. *Generate and push the migration.*

Before continuing, confirm the webhook URL is a valid-looking Slack URL and not any other URL I might have pasted by mistake. (Regex check: it must start with `https://hooks.slack.com/services/` *.)*

VERIFY **Step 3 · Verify:**

- Cursor confirms the webhook URL format is correct.
- `contact_submissions` table exists in Neon (Cursor queries `information_schema` to prove it).
- `SLACK_CONTACT_WEBHOOK_URL` is in `.env.local` and Vercel's prod + preview.

PROMPT **Step 4 · Say to Cursor:**

Build a `/contact` *page and its server action:*

UI (`/contact` ***):***

- *Clean form: Name, Email, Message textarea,* ***Send*** *button. Match the site's design tokens and accent colour.*
- *Real-time character count on the message field (10 min / 2000 max).*
- *A visible* ***hidden honeypot field*** *named* `website` *— label it something boring like* ***Your website (optional)****, hide it visually with CSS (not with* `hidden` *), and the server rejects any submission where this is filled.*
- *After submit: show a success state* ***"Thanks! I'll be in touch within 48 hours."*** *Don't navigate away.*

Server action (`app/contact/actions.ts` ***):***

- *Validate inputs with zod. Reject honeypot-filled submissions silently (return success UI but don't persist and don't notify — this confuses bots).*
- *Hash the request IP (SHA-256) and extract* `user-agent` *; store both.*
- *INSERT into* `contact_submissions` *.*
- *Fire the Slack webhook via* `fetch(POST)` *with a message that looks like:*

  > ***New contact form submission*** > ***From:*** *[name] (* `\[email\]` *)* > ***Message:*** *[message, first 400 chars]* > received [ISO timestamp]
- *If the Slack call fails, still persist the DB row and log the failure — never drop the user's message because Slack hiccupped.*
- *Return an appropriate success / error shape to the form UI.*

VERIFY **Step 5 · Verify:**

- Localhost: submit the form with valid data. Success state appears.
- A Slack message arrives in `#my-site-contact` within ~2 seconds.
- The row is in Neon.
- Submit with the honeypot filled — form shows success state but nothing arrives in Slack or Neon (silent drop).
- Submit with message length 5 — form shows validation error, no submission.

PROMPT **Step 6 · Say to Cursor:**

Add two cheap spam defences:

1. ***Rate limit per IP hash:*** *5 submissions per hour. Use an in-memory LRU (simple Map with TTL) since we haven't set up Upstash — that's fine for class. When exceeded, return a 429 and show* ***"You've sent a lot of messages recently — try again in an hour."***
2. ***Content heuristic:*** *if the message contains more than 3 URLs, or if the message is identical to the previous submission in the last 10 minutes, silently drop it (same UX as honeypot).*

Add tiny Slack notifications for each drop reason to a separate `#my-site-spam` *channel if I set up a second webhook, otherwise just log server-side.*

VERIFY **Step 7 · Verify:**

- Submit 6 times in a row from your browser — 6th is rate-limited.
- Submit the same message twice within 10 minutes — second is silently dropped.
- Submit a message with 5 URLs — silently dropped.

PROMPT **Step 8 · Say to Cursor:**

Commit everything, push, wait for the Vercel deploy. Once live, submit a test form from the production URL yourself to verify the Slack webhook works in prod (env vars sometimes only get added to `preview` *by mistake). Also, add a small* ***Contact*** *link to the site header next to* ***Blog****.*

VERIFY **Step 9 · Verify:**

- Header now has **Blog** · **Contact** next to existing nav items.
- A production submission arrives in Slack within 2 seconds.
- The Neon row shows a hashed IP (not raw).

RECOVER **Step 10 · If stuck, say to AI:**

Slack messages stopped arriving 20 minutes ago, but the form still returns success. Please check the Vercel function logs for the contact action, identify where the Slack POST is failing, and walk me through fixing it. Prioritise: do not start storing users' messages ***without*** *Slack delivery unless we explicitly decide that's OK.*

WARNING · Your webhook URL is a password-equivalent

> Anyone with your Slack webhook URL can post anything into your channel, including spam or abusive content. **Never** commit it to GitHub, never paste it into public screenshots, never send it in a DM as "here's a link". If it ever leaks, rotate it in the Slack app settings immediately — Cursor can update your `.env` + Vercel env vars in one prompt once you have the new URL.

Weekly Assignment

Build / Implement.

- Live `/contact` page that saves to Neon and notifies your Slack.
- Rate limit + honeypot + duplicate-content defences working.

Requirements.

- Honeypot field invisible in the normal UI but present in the DOM.
- IPs are hashed, never stored raw.
- Live submission from a classmate's computer triggers a Slack ping within 3 seconds.
- Screenshot of your Slack channel showing at least three classmate submissions.

Submission. Live URL + Slack screenshot in the course channel before Week 8.

Resources

Docs

- Slack Incoming Webhooks — docs
- Slack Block Kit — for nicer messages (optional)
- `zod` — schema validation
- Next.js server actions — docs

Videos

- Instructor demo: "Real-time Slack pings in 12 minutes"

Repos

- `vercel/next.js/examples/with-server-actions`

Real-World Application

Every indie product and early-stage startup begins its observability journey the same way: **a Slack channel with automated pings for every important event**. Before Grafana, before Datadog, before PagerDuty — just one Slack channel called `#ops` or `#events`. Getting this habit into your fingers now means that when you ship your first production product, you'll already know how to feel its pulse without spending a cent on tooling.

> CAREER
>
> After class, add a second webhook for guestbook posts (cheap 10-minute prompt). Now you're notified when anyone signs in and leaves a message on your site — and you've experienced the feeling of **my product is alive and real people are using it**. That feeling is addictive. Protect it by never letting your Slack channel become so noisy you mute it.

Challenges & Tips

- **"Slack says my webhook URL is invalid."** You probably copied with a trailing space, or grabbed the docs URL instead of your specific webhook. Paste into Cursor and ask it to verify the format.
- **"Messages arrive in Slack but look ugly."** Default plain-text is fine for now, but Slack's Block Kit builder generates prettier JSON. Ask Cursor **"Switch the Slack message to Block Kit with a coloured sidebar, fields for each value, and a muted timestamp."**
- **"Rate limit counts every submission twice."** Client revalidation fired twice. Ask Cursor **"Debounce the form submission handler and ensure the server action only counts once per actual insert."**
- **"On Vercel the timezone in Slack is wrong."** Ask **"Format all timestamps as user-local time in `Asia/Shanghai` or pass an explicit IANA TZ."**
- **"I'm getting spam already."** Good — real bots found you, which means your site is indexable. Tighten: shorten the rate-limit window to 2 per hour, add a keyword blocklist.

> **STUCK? RESCUE**
>
> If the form silently succeeds but nothing arrives in Slack **and** nothing is in Neon, the honeypot field is probably visible to humans too. Ask Cursor **"Check the computed styles of my `website` honeypot field — is it actually hidden visually?"**

Instructor Notes (Internal)

- **Project the instructor's Slack channel during the demo.** The wow moment is seeing the ping arrive in real time. Don't skip this — it's why the class is called **COOL**.
- **Spam-defence layer is a safe place to geek out.** Students who like security will love the content-heuristic step. Don't cap it at 3 URLs — let them design their own rules and compare.
- **Webhook URLs leaking.** One student will inevitably commit the webhook URL to GitHub by pasting into a `.md` file and then `git add .`. Have the rotation drill ready: the fix takes 90 seconds with Cursor if you know the pattern.
- **Pair students up for testing.** Each student submits once on three classmates' forms. Everyone gets real submissions in their Slack, and the class has a tiny social moment.
- **Next week is the grand finale.** Tease Week 8: **"Everything you've built this term becomes the data for a real CV and cover letter generator."**

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.