

SKILLNAVORA · AI PRODUCT MANAGEMENT & GENAI
LAUNCHPAD · AI TOOLS

Web Automation: Webhooks, Triggers & Playwright MCP

The modern web-automation stack in one course:
webhooks, triggers (schedules, events, polling), browser
automation with Playwright, the Playwright MCP that lets
Claude and other AI assistants drive a real browser, and
every major alternative: Selenium, Puppeteer, Cypress,
Chrome DevTools MCP, Browser Use, Stagehand, Skyvern,
Browserbase, Apify, n8n, Zapier, Make, Pipedream, RPA and
AI browsers.

Edition 2026 · skillnavora.in

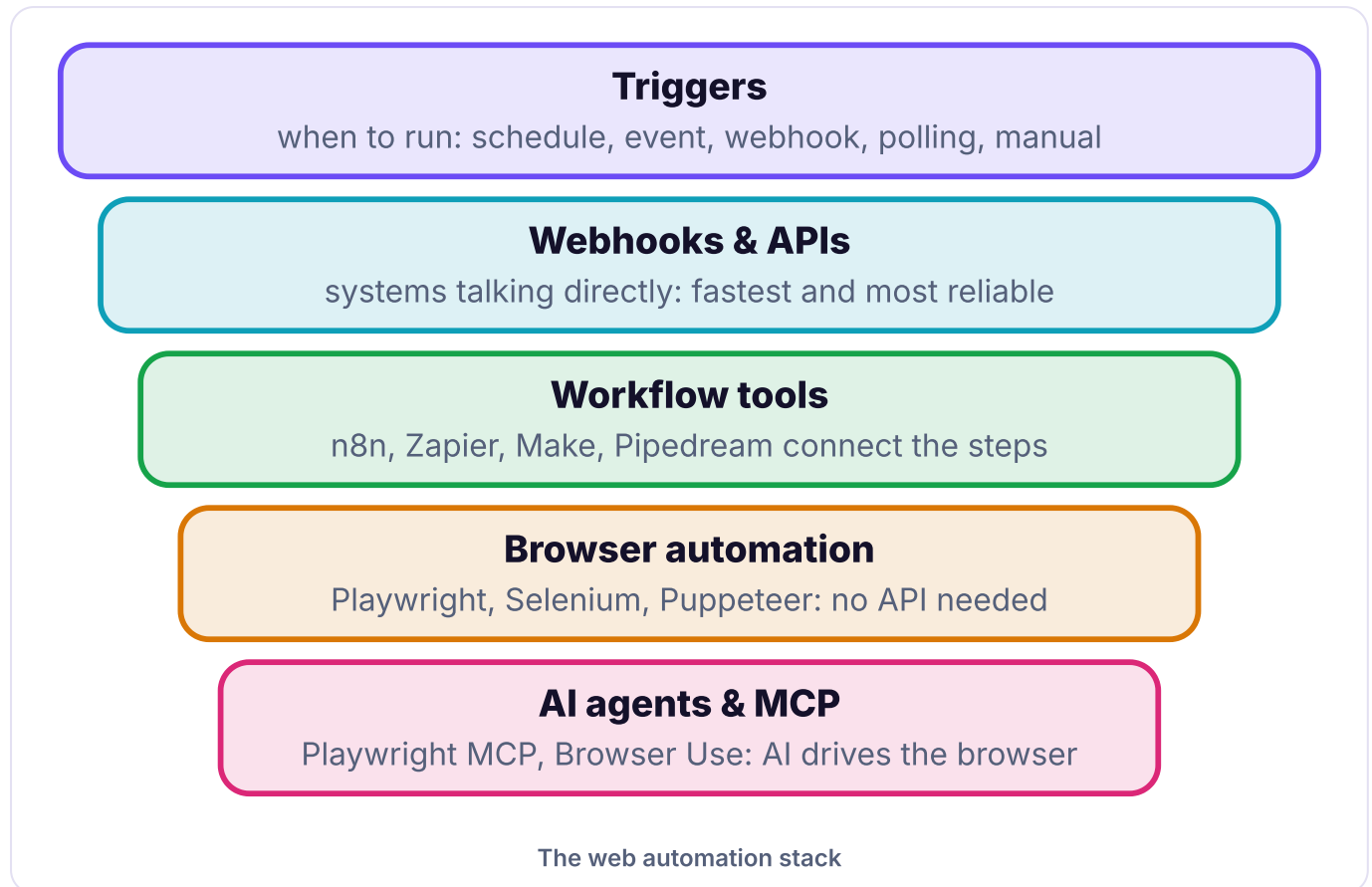
How to use this guide

Each class has goals, concepts with Indian examples, key takeaways and a practice exercise, from basic to advanced. Finish with the final project to build something you can show in interviews.

	Class	Level
Class 1	The web-automation stack: APIs, webhooks, triggers, browsers	Basic
Class 2	Webhooks explained (with real examples)	Basic
Class 3	Building reliable webhook receivers	Intermediate
Class 4	Triggers: schedules, events, polling and manual	Basic
Class 5	Playwright for browser automation	Intermediate
Class 6	Playwright MCP: let Claude and other AI assistants use a browser	Intermediate
Class 7	Other browser-automation frameworks and MCPs	Intermediate
Class 8	AI browser agents and cloud browsers	Advanced
Class 9	Workflow platforms: n8n, Zapier, Make, Pipedream, RPA and scrapers	Advanced
Class 10	Production: security, legality, monitoring, evals and capstone	Advanced
Toolkit: templates, platform cheat sheet, glossary, 10 interview questions, further reading		

The web-automation stack: APIs, webhooks, triggers, browsers

See how the pieces fit, and pick the right one for each job.



Situation	Use
Another app should tell you instantly when something happens (payment, form, new order)	Webhook
You need data or actions from a service that has an API	API call
Something should run every hour / day / Monday	Scheduled trigger (cron)
A website has no API and you must click through it	Browser automation (Playwright)
You want an AI assistant to research or test websites for you	Playwright MCP or an AI browser agent

Golden rule: webhook or API first; browser automation only when there's no API; AI agents for messy, changing tasks; humans approve anything irreversible.

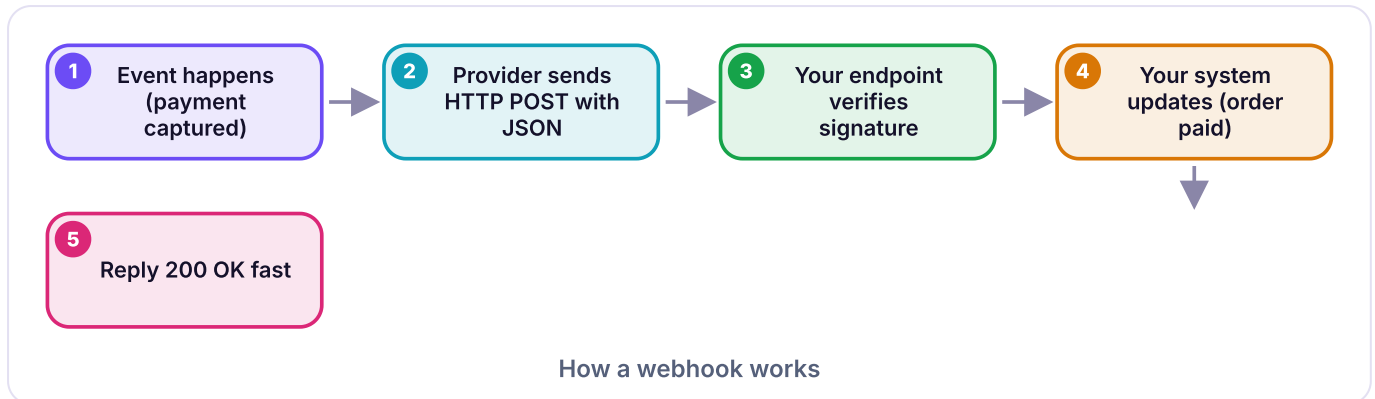
Key takeaways

- Triggers decide when; webhooks and APIs connect systems; browsers click when nothing else works.
- Prefer APIs and webhooks for reliability.
- AI agents sit on top for flexible tasks.

Practice exercise: Map 5 tasks from your work to webhook, API, schedule, browser automation or AI agent.

Webhooks explained (with real examples)

Understand webhooks and receive your first one.



A **webhook** is an automatic message one app sends to a URL you give it when an event happens. Instead of asking "any new payments?" every minute (polling), the provider *pushes* the news to you instantly.

Common webhooks

Provider	Example events
Razorpay / Stripe	payment.captured, refund.processed, subscription.charged
GitHub	push, pull_request, issues
Typeform / Google Forms (via tools)	new response
Shopify / WooCommerce	orders/create, orders/paid
Slack / WhatsApp Business	message received
Calendly	invitee.created

Example payload

```

POST https://yourapp.com/webhooks/razorpay
{ "event": "payment.captured",
  "payload": { "payment": { "entity": { "id": "pay_29QQoUBi66xm2f", "amount": 49900,
    "currency": "INR", "email": "riya@example.com", "status": "captured" } } } }
  
```

Try it in 5 minutes (no code)

1. Open a free webhook tester (e.g. webhook.site) and copy your unique URL.
2. Paste it as the webhook URL in a test tool (a form tool, GitHub repo settings → Webhooks, or Razorpay test mode).
3. Trigger the event (submit the form, push a commit) and watch the request arrive with its headers and JSON.

Key takeaways

- Webhooks push events to you in real time.
- The body is usually JSON describing the event.
- Test with a webhook inspector before writing code.

Practice exercise: Receive a real webhook from GitHub or a form tool in a webhook tester and identify the event name and key fields.

Building reliable webhook receivers

Verify, process and retry webhooks safely in production.

Six rules for production webhooks

1. **Verify the signature** (HMAC) so only the real provider can call you.
2. **Respond fast** (200 OK within a few seconds); do heavy work in a queue.
3. **Be idempotent**: providers retry, so the same event may arrive twice; store event IDs and skip duplicates.
4. **Handle retries and order**: events can arrive late or out of order; check current state.
5. **Log everything**: raw payload, headers, result; keep a replay option.
6. **Secrets in environment settings**, never in code.

Signature check (Node.js / Cloudflare-style function)

```
import crypto from "node:crypto";
export async function onRequestPost({ request, env }) {
  const body = await request.text(); // raw body, before JSON parsing
  const sig = request.headers.get("x-razorpay-signature") || "";
  const expected = crypto.createHmac("sha256",
    env.RAZORPAY_WEBHOOK_SECRET).update(body).digest("hex");
  if (sig.length !== expected.length ||
    !crypto.timingSafeEqual(Buffer.from(sig), Buffer.from(expected)))
    return new Response("invalid signature", { status: 401 });
  const event = JSON.parse(body);
  // idempotency: skip if this event id was already processed (check your DB/KV)
  // enqueue the work, then reply quickly
  return new Response("ok");
}
```

Each provider documents its own header name and signing method; follow its official guide.

Local testing

Expose your laptop with a tunnel (ngrok, Cloudflare Tunnel) or use the provider's CLI to forward test events; replay saved payloads in tests.

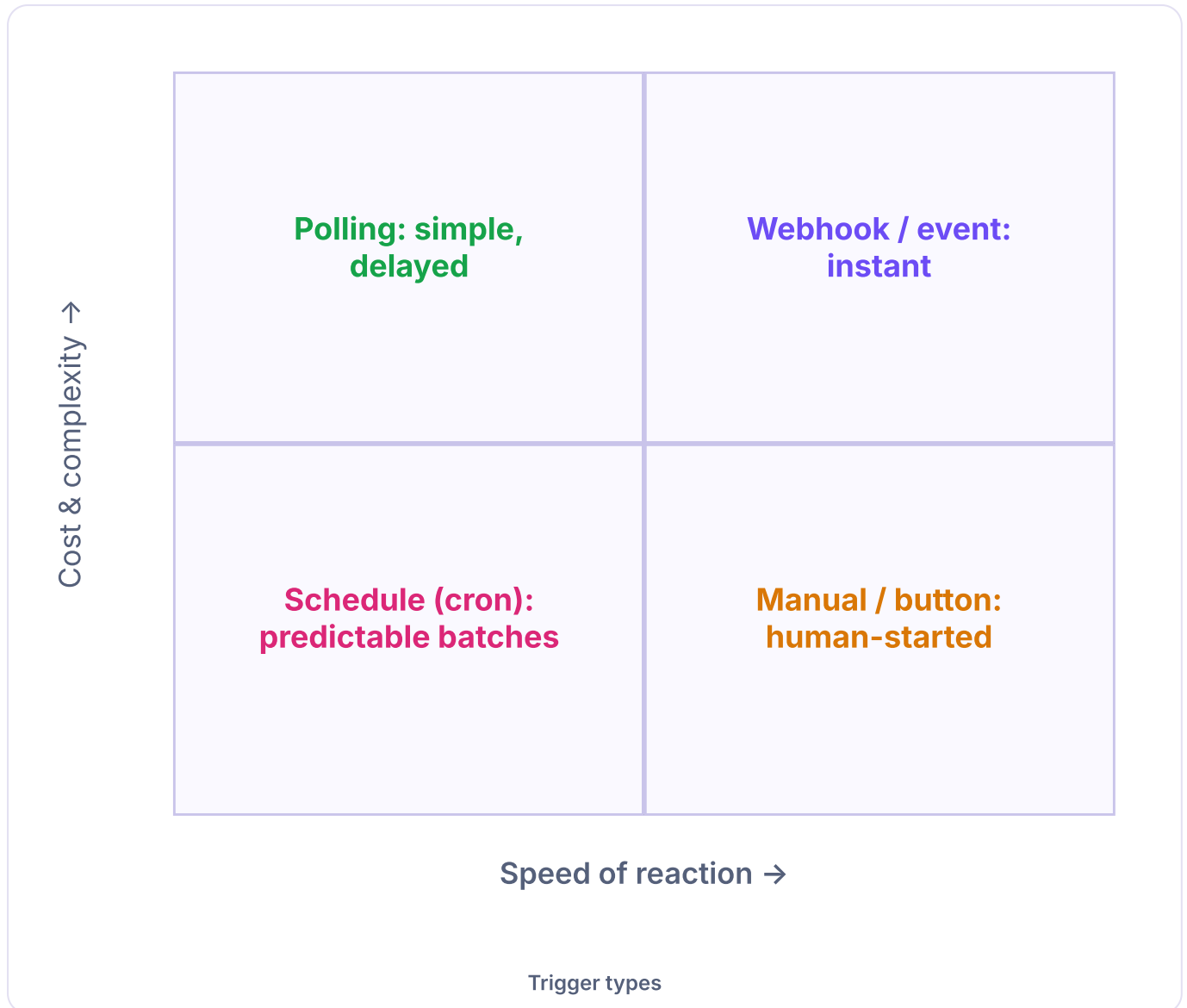
Key takeaways

- Verify signatures, respond fast, be idempotent.
- Expect retries and out-of-order events.
- Log and replay for debugging.

Practice exercise: Write the requirements (or code) for a webhook receiver that marks orders paid, including signature, idempotency and logging.

Triggers: schedules, events, polling and manual

Choose how and when automations start.



Trigger	Example	Where
Schedule (cron)	Every day 9:00 IST: download sales report	n8n Schedule, Zapier Schedule, Make scheduling, GitHub Actions cron, Cloudflare Cron Triggers
Webhook / event	New payment → send receipt on WhatsApp	Any webhook trigger node
Polling	Every 15 min check a Google Sheet for new rows	Most "new row/new email" app triggers
App event	New Gmail with label "lead"	Zapier/Make/n8n app triggers
File / folder	New CSV in Drive → import	Drive/Dropbox triggers
Manual / chat	Button in Slack or a form; "run now"	Slack commands, forms, chat triggers

Cron in one minute

```
minute hour day-of-month month day-of-week
0 9 * * 1-5      → 09:00 every weekday
*/15 * * * *     → every 15 minutes
30 18 1 * *      → 18:30 on the 1st of every month
Note: many servers use UTC; 09:00 IST = 03:30 UTC.
```

Key takeaways

- Webhooks for instant, schedules for batches, polling when nothing else exists.
- Watch time zones in cron (IST vs UTC).
- Add manual triggers for testing.

Practice exercise: Write cron expressions for 3 schedules you need, in IST and UTC.

Playwright for browser automation

Automate any website reliably with Playwright.

Playwright (by Microsoft) controls Chromium, Firefox and WebKit from Python, JavaScript/TypeScript, Java or .NET. It auto-waits for elements, records your clicks as code and replays failures step by step.

Setup and record

```
pip install playwright && playwright install chromium
playwright codegen https://demo.playwright.dev/todomvc/ # click around; code is written
for you
```

Script: log in, export a report (pattern)

```
from playwright.sync_api import sync_playwright
with sync_playwright() as p:
    browser = p.chromium.launch(headless=True)
    ctx = browser.new_context(storage_state="auth.json") # reuse a saved, authorised login
    page = ctx.new_page()
    page.goto("https://portal.example.com/reports")
    page.get_by_label("From").fill("2026-10-01")
    page.get_by_role("button", name="Export CSV").click()
    with page.expect_download() as d:
        page.get_by_role("link", name="Download").click()
    d.value.save_as("report.csv")
    browser.close()
```

Make it robust

- Locate by role, label, text or test id, not fragile CSS paths.
- Use `storage_state` for logins (store securely; never commit it).
- Turn on tracing; open failures in the trace viewer.
- Run on a schedule (GitHub Actions cron, n8n Execute Command, a small server) with alerts on failure.

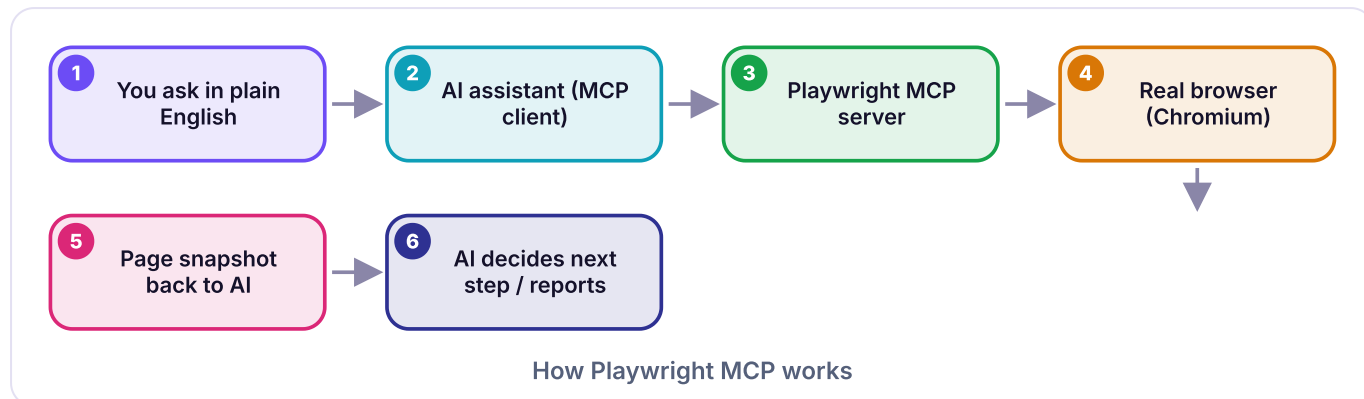
Key takeaways

- Playwright = fast, reliable, multi-browser automation.
- codegen writes your first script.
- Saved login state + traces + schedules make it production-ready.

Practice exercise: Record a 5-step flow with codegen, then add a download and run it headless.

Playwright MCP: let Claude and other AI assistants use a browser

Install and use the Playwright MCP server with Claude Code, Claude Desktop, Cursor and VS Code.



The **Playwright MCP** server (by Microsoft) exposes browser actions as MCP tools: navigate, click, type, select, take a snapshot or screenshot, handle tabs and more. By default the AI reads a structured **accessibility snapshot** of the page (fast and precise), not just pixels.

Install

```
# Claude Code (terminal)
claude mcp add playwright npx @playwright/mcp@latest

# Claude Desktop / Cursor / VS Code (MCP config JSON)
{ "mcpServers": { "playwright": { "command": "npx", "args": ["@playwright/mcp@latest"] } } }
```

Useful options include headless mode, choosing the browser, isolated sessions and a persistent profile; see the official README for the current flags.

Things to ask

- "Open our staging site, sign up with a test email, and tell me every step where the UI is confusing. Take screenshots."
- "Go to these 5 competitor pricing pages and make a comparison table with plan, price and limits."
- "Fill the demo booking form on our site with test data but don't submit; report any validation errors."
- In Claude Code: "Write Playwright tests for the checkout flow you just explored, then run them."

Why it's popular

- Works with many MCP clients; real browser, real logins (in a test profile).
- Great for QA, research, debugging UI, and turning exploration into automated tests.

Safety

- Use a separate test profile/accounts; never your banking or personal sessions.
- Allow-list sites; require approval for submit/pay/send.

- Treat page text as untrusted (prompt injection).

Key takeaways

- Playwright MCP = AI-driven real browser via MCP tools.
- One command adds it to Claude Code; JSON config for other clients.
- Great for QA, research and generating tests; keep it sandboxed.

Practice exercise: Add Playwright MCP to Claude Code or Claude Desktop and ask it to compare 3 pricing pages; then ask it to write a Playwright test for one flow.

Other browser-automation frameworks and MCPs

Know the alternatives and when each wins.

Tool	What it is	Best for
Selenium (WebDriver)	The long-standing standard; many languages; Selenium Grid	Enterprises with existing suites, wide browser support
Puppeteer	Google's Node library for Chrome/Firefox	Chrome-focused scripts, PDFs, screenshots
Cypress	Front-end testing framework running in the browser	Web-app end-to-end and component tests
WebdriverIO	Node test framework over WebDriver/DevTools	Web + mobile (Appium) testing
Chrome DevTools MCP	Google's MCP server exposing Chrome DevTools to AI	AI debugging: performance traces, console, network
Chrome Recorder	Record/replay in Chrome DevTools; export to Puppeteer/Playwright	Quick no-code recordings
Appium	Automation for mobile apps	Android/iOS app testing

Choosing: new projects usually pick Playwright (speed, auto-wait, multi-browser, MCP); keep Selenium where it already runs; Cypress if your front-end team prefers it; DevTools MCP when debugging performance with AI.

Key takeaways

- Playwright is the default for new work.
- Selenium remains common in enterprises.
- Pick by team skills and browsers needed.

Practice exercise: Compare Playwright, Selenium and Cypress for a QA plan of your product's sign-up flow.

AI browser agents and cloud browsers

Use AI-native automation tools and hosted browsers at scale.

Tool	What it does
Browser Use (open source)	Python library: give an LLM a goal; it plans and drives a browser
Stagehand (by Browserbase)	Playwright plus AI actions (act, extract, observe) in plain English inside code
Skyvern	AI agent for form-heavy workflows using vision + DOM
Browserbase, Browserless, Steel	Cloud browser infrastructure: run many headless browsers, sessions, proxies, recordings
Claude in Chrome / computer use	Claude operating your browser or a computer with permission
ChatGPT agent mode	OpenAI's agent completing web tasks in a virtual browser
Perplexity Comet and other AI browsers	Browsers with built-in assistants that act on pages

Pattern: AI where needed, code where possible

Playwright script for stable steps → AI action (Stagehand/Browser Use) for the messy page → validation → human approval → save results

Measure success rate, cost per task and safety (see Evals and Browser Automation courses).

Key takeaways

- AI agents handle changing pages; code handles stable steps.
- Cloud browsers scale and isolate sessions.
- Always measure reliability and cost.

Practice exercise: Design a hybrid workflow (script + AI step) for collecting prices from 10 sites.

Workflow platforms: n8n, Zapier, Make, Pipedream, RPA and scrapers

Run webhooks, triggers and browser steps as complete workflows.

Platform	Strengths
n8n	Webhook and schedule triggers, AI agent nodes, code nodes, self-hostable; can call Playwright scripts or browser services
Zapier	Easiest; "Webhooks by Zapier" catch and send hooks; huge app library
Make	Visual scenarios, custom webhooks, good value
Pipedream	Developer-friendly: webhook triggers + code steps (Node/Python)
IFTTT	Simple personal automations
Apify	Ready "Actors" for scraping and browser automation, schedules, datasets, API
Bardeen, Axiom	Browser-extension automation for repetitive web tasks
UiPath, Power Automate, Automation Anywhere	Enterprise RPA for desktop + web with governance

Example: payment webhook → n8n → browser step → WhatsApp

1. Webhook node receives Razorpay payment . captured (verify signature).
2. Look up the order in Google Sheets/DB.
3. If the partner portal has no API, call a Playwright script (or Apify actor) to register the booking.
4. Send a WhatsApp/email confirmation; log the run; alert on failure.

Key takeaways

- n8n/Zapier/Make/Pipedream orchestrate triggers and actions.
- Apify and RPA tools cover scraping and enterprise desktop work.
- Combine webhook + workflow + browser step + notification.

Practice exercise: Build the payment-to-confirmation workflow in n8n or Make with a test webhook.

Production: security, legality, monitoring, evals and capstone

Run web automations safely and prove they work.



Checklist

- Secrets in environment settings; rotate keys; least-privilege accounts.
- Respect site terms and robots.txt, rate-limit, never bypass CAPTCHAs; prefer APIs; follow the DPDP Act for personal data.
- Approvals before submit/pay/send/delete; allow-listed sites for AI agents.
- Monitoring: success rate, duration, cost, failures with screenshots/traces; alerts to Slack/email.
- Evals: 20–50 test tasks with known end states; run after changes (see AI Evals course).

Capstone

Build a "competitor & order watch": (1) a webhook receiver for test payments with signature check and idempotency, (2) a daily cron workflow in n8n, (3) a Playwright script that collects 3 competitor prices, (4) a Playwright MCP session in Claude that explores your own site and writes one automated test, (5) alerts and a run log, (6) a 1-page PRD with an approval matrix. Demo in 5 minutes.

Key takeaways

- Secure, legal, approved, monitored, evaluated.
- Every failure leaves a trace and an alert.
- Ship a capstone you can demo in interviews.

Practice exercise: Complete the capstone and present it.

Templates

A. Webhook spec

Field	Fill in
Provider & events	
Endpoint URL	
Signature method & secret name	
Idempotency key	
Processing (queue?)	
Retries & dead-letter	
Logging & alerts	

B. Trigger plan

workflow | trigger type | schedule/event | time zone | owner | failure alert

C. Playwright MCP session brief

Goal:
Sites allowed:
Test account / profile:
Do NOT: submit / pay / send / delete
Output: table / findings / screenshots / generated test file

Platform cheat sheet

Term / tool	Meaning / use
Webhook	Event pushed to your URL in real time
Trigger	What starts an automation: schedule, event, polling, manual
Cron	Schedule syntax: minute hour day month weekday
Playwright	Modern browser automation & testing (Microsoft)
Playwright MCP	MCP server so AI assistants drive a real browser
Chrome DevTools MCP	AI access to Chrome DevTools for debugging
Selenium	Long-standing WebDriver standard
Puppeteer	Chrome automation for Node
Cypress	Front-end end-to-end testing
Browser Use / Stagehand / Skyvern	AI-native browser automation
Browserbase / Browserless / Steel	Cloud browsers at scale
Apify	Scraping & automation Actors
n8n / Zapier / Make / Pipedream	Workflow platforms with webhook & schedule triggers
UiPath / Power Automate	Enterprise RPA

Glossary

HMAC	Signature proving a message came from the sender
Payload	The data sent in a request
Endpoint	URL that receives requests
Idempotency	Same request twice = same result
Headless	Browser without a visible window
Accessibility snapshot	Structured view of a page's elements used by Playwright MCP
Tunnel	Exposes a local server to the internet for testing

10 interview questions with answers

Q1. What is a webhook?

An HTTP request an app sends to your URL when an event happens, so you get updates instantly instead of polling.

Q2. How do you secure a webhook endpoint?

Verify the provider's HMAC signature on the raw body, use HTTPS, keep the secret in environment settings, respond fast, and process idempotently.

Q3. Webhook vs polling?

Webhooks push instantly and save calls; polling checks repeatedly and adds delay but works when no webhooks exist.

Q4. What is idempotency and why does it matter?

Processing the same event twice has no extra effect; needed because providers retry and may send duplicates.

Q5. What is the Playwright MCP?

An MCP server by Microsoft that exposes browser actions (navigate, click, type, snapshot) so AI assistants like Claude can use a real browser.

Q6. Playwright vs Selenium?

Playwright: newer, faster, auto-waiting, multi-browser, great tooling and MCP. Selenium: mature standard with broad language and grid support, common in existing enterprise suites.

Q7. When would you use an AI browser agent instead of a script?

When pages change often or tasks vary; keep scripts for stable steps and add AI for the messy parts, with evals and approvals.

Q8. How do you schedule an automation for 9 am IST on weekdays?

Cron '0 9 * * 1-5' in IST, or '30 3 * * 1-5' if the server runs in UTC.

Q9. What risks come with browser automation?

Breaking on UI changes, terms-of-service and legal issues, credential leaks, prompt injection for AI agents, and irreversible actions; mitigate with APIs, sandboxes, approvals and monitoring.

Q10. Give a PM use case for Playwright MCP.

Ask Claude to walk the onboarding flow on staging, list UX issues with screenshots, then generate Playwright tests for the happy path.

Further reading and tools

- Playwright docs: <https://playwright.dev/docs/intro>
- Playwright MCP (GitHub): <https://github.com/microsoft/playwright-mcp>
- Chrome DevTools MCP (GitHub): <https://github.com/ChromeDevTools/chrome-devtools-mcp>
- Selenium docs: <https://www.selenium.dev/documentation/>
- Puppeteer docs: <https://pptr.dev/>
- Cypress docs: <https://docs.cypress.io/>
- Browser Use: <https://github.com/browser-use/browser-use>
- Stagehand: <https://www.stagehand.dev/>
- Skyvern: <https://www.skyvern.com/>
- Browserbase: <https://www.browserbase.com/>
- Apify docs: <https://docs.apify.com/>
- n8n webhook node: <https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.webhook/>
- Zapier webhooks: <https://zapier.com/apps/webhook/integrations>
- Pipedream docs: <https://pipedream.com/docs>
- Razorpay webhooks: <https://razorpay.com/docs/webhooks/>
- GitHub webhooks: <https://docs.github.com/en/webhooks>
- Crontab guru (cron helper): <https://crontab.guru/>
- Model Context Protocol: <https://modelcontextprotocol.io/>

© SkillNavora · skillnavora.in · Part of the AI Product Management and GenAI Launchpad programs. Practise with the free AI Tutor, AI PM Toolkit and Practice Lab on skillnavora.in.