DeltaForceOS

CODEX ROUTE / BEGINNER MODULE 06

Beginner Web Scraper Agent: Codex Route

BUILD OUTCOME

Build a scraper that reads pages, extracts structured data, dedupes results, and fails safely.

BEST FOR

Use this PDF when you want exact prompts and clicks for this route instead of general theory.

MODEL TIER

Light for extraction; mid for scoring; no heavy model until the parser is stable.

FIRST INSTRUCTION

```
Inspect this repo for the Beginner Web Scraper Agent build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

Before using the agent, make the workspace boring and clear.

01 Open the project folder. If you do not have one yet, create a new folder on Desktop and name it after the lesson.

02 Install Node.js if npm is not available. Beginners can check by opening Terminal and typing npm -v.

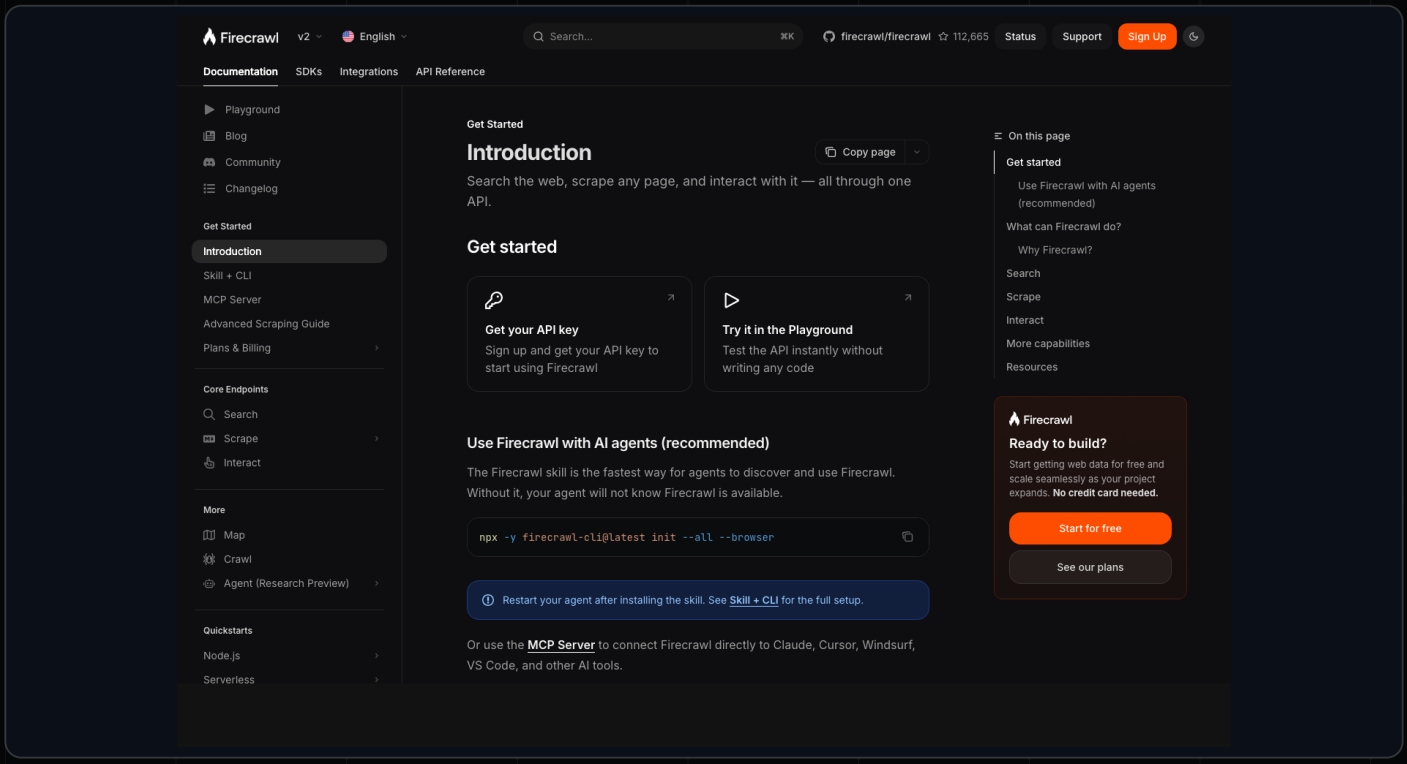
03 Create .env.local in the project root and add only the keys this lesson needs.

04 Confirm .env.local is listed in .gitignore so secrets do not go to GitHub.

05 Open the official dashboard for the first two tools in this lesson and create one tiny test project.

06 Decide where proof will live: Supabase row, screenshot, live URL, or Skool post.

Firecrawl: click path for this route.



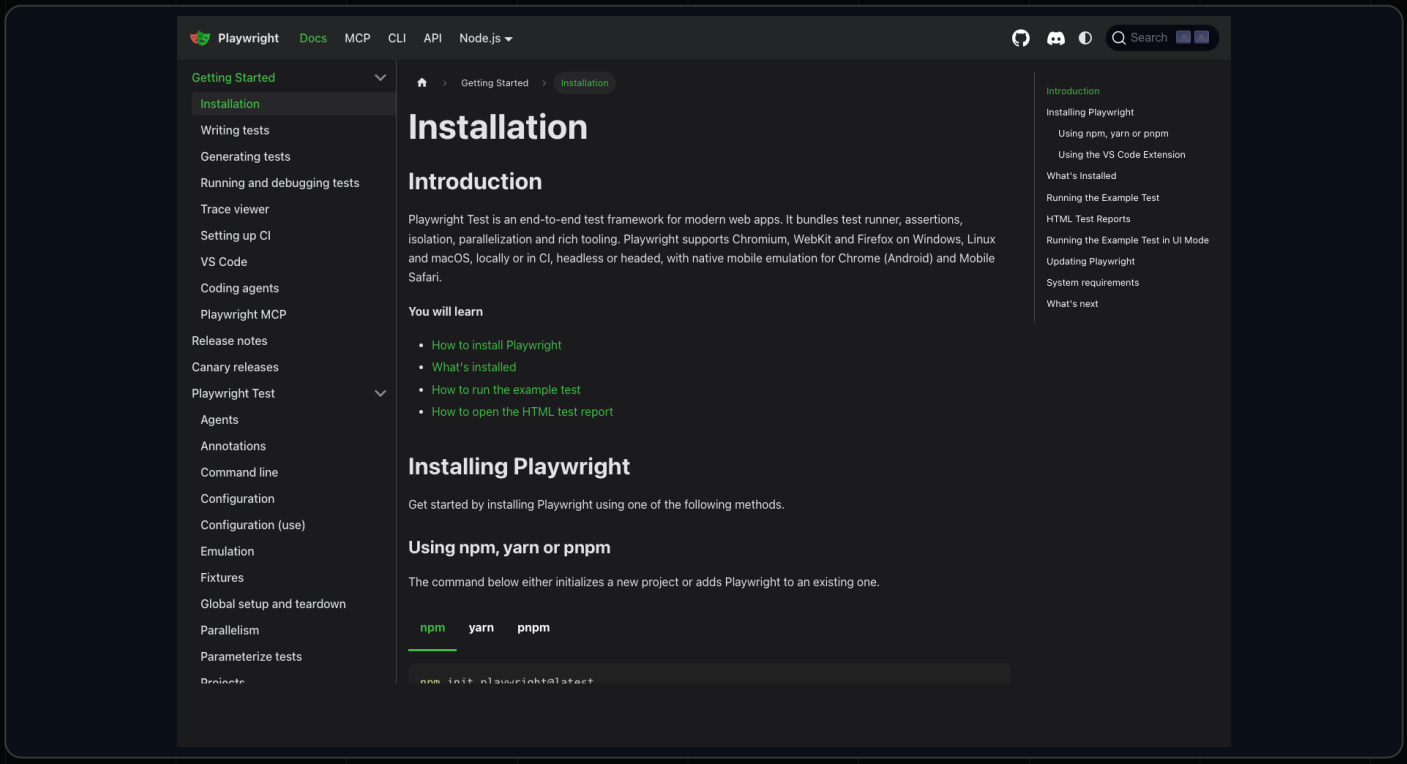
WHERE TO CLICK

Open Firecrawl, create an API key, test one target in the playground, then call scrape/search from a worker.

ENV NAMES

```
FIRECRAWL_API_KEY
```

Playwright: click path for this route.



WHERE TO CLICK

Install Playwright, run one generated test, then use traces and screenshots to debug.

ENV NAMES

```
Usually none; use TEST_BASE_URL for deployed preview tests.
```

Copy this box exactly, then let the agent ask clarifying questions only if needed.

INSPECT PROMPT

```
Inspect this repo for the Beginner Web Scraper Agent build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

Copy this box exactly, then let the agent ask clarifying questions only if needed.

BUILD PROMPT

```
Implement the smallest useful version of Beginner Web Scraper Agent.
Outcome: Build a scraper that reads pages, extracts structured data, dedupes results, and fails safely.
Tools allowed: Firecrawl, Playwright, Supabase, OpenAI, Python, Lighthouse, Vercel
Keep secrets server-side. Add code comments only where a beginner would get lost.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

TYPE THIS INTO FILES

```
# README.md lesson block
Lesson: Beginner Web Scraper Agent
Outcome: Build a scraper that reads pages, extracts structured data, dedupes results, and fails safely.
Tools: Firecrawl, Playwright, Supabase, OpenAI, Python, Lighthouse, Vercel

# .env.local placeholders
NEXT_PUBLIC_SUPABASE_URL=your_value_here
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_value_here
OPENAI_API_KEY=server_only
ANTHROPIC_API_KEY=server_only

# Verify
npm run build
```

WHERE IT GOES

README.md gets instructions. .env.local gets real local values. Vercel gets production values. Source files get implementation.

Basic English. Click by click. Do not skip ahead.

01 Open chrome and go to firecrawl.dev. Click Sign up. Sign in with Google or GitHub.

02 On the dashboard, click API Keys. Copy your Firecrawl key into a sticky note.

03 Open .env.local in your editor. Add FIRECRAWL_API_KEY=your-key. Save.

04 Open Terminal in your project. Type npm install @mendable/firecrawl-js and press Enter.

05 Open Supabase Table editor. Click + New table named scraped_pages with columns: id (uuid primary), url (text unique), title (text), markdown (text), scraped_at (timestamp default now()).

06 Create a new file src/app/api/scrape/route.ts. Paste: import FirecrawlApp from '@mendable/firecrawl-js'; export async function POST(req) { const { url } = await req.json(); const fc = new FirecrawlApp({ apiKey: process.env.FIRECRAWL_API_KEY }); const r = await fc.scrapeUrl(url, { formats: ['markdown'] }); return Response.json(r); }

Finish the build. The last step always posts proof in Skool.

01 Save the file. In Terminal, run: curl -X POST http://localhost:3000/api/scrape -H 'Content-Type: application/json' -d '{"url":"https://example.com"}'

02 You should see JSON with clean markdown. If you see 401 unauthorized, your Firecrawl key is wrong.

03 Add a Supabase insert inside route.ts that saves url, r.metadata.title as title, and r.markdown into scraped_pages.

04 Pick five real public sites you want to audit. Create a file urls.txt with one URL per line.

05 In Terminal, run: while IFS= read -r url; do curl -X POST http://localhost:3000/api/scrape -H 'Content-Type: application/json' -d "{\"url\":\"$url\"}"; done < urls.txt

06 Open Supabase Table editor and the scraped_pages table. Confirm five rows. Take a screenshot for the Beginner 06 Skool thread.

The build is not finished until verification is boring.

01 Can it survive five messy websites?

02 Can you re-run extraction without re-crawling?

03 Can a human audit the source behind each field?

04 Run npm run build or the focused test command.

05 Download or open the produced artifact and inspect it visually.

06 Write the failure and fix in README.md or the Skool proof post.

01 Commit safe files to GitHub after reviewing the diff.

02 Open Vercel and deploy a preview first.

03 Add missing env vars under Project Settings -> Environment Variables if the build fails.

04 Run the workflow with fake or test data in preview.

05 Promote or deploy production only after the smoke test passes.

06 Save the live URL and proof artifact in Skool.

AGENT DEPLOY PROMPT

```
Verify the preview build, list any missing env vars, and give me the exact smoke test before production.
```

If something goes wrong, ask for a small repair instead of a total rewrite.

01 Scraping private or disallowed pages.

02 Letting one broken website crash the whole batch.

03 Only saving the AI summary and losing the raw source.

04 Ask the agent to explain the failing layer: local app, env var, database, external API, prompt, deploy, or auth.

05 Ask for one fix, one test, and one rollback step.

06 Do not let the agent change unrelated files to hide the failure.

SKOOL PROOF

Scrape five public websites and produce one structured audit row per site.

WHAT TO POST

Artifact, screenshot or URL, what worked, what broke, exact prompt used, and the next target.

SOURCES

Codex: https://developers.openai.com/codex

Firecrawl: https://docs.firecrawl.dev/introduction

Playwright: https://playwright.dev/docs/intro

Supabase: https://supabase.com/docs/guides/getting-started

OpenAI: https://platform.openai.com/docs/quickstart

Python: https://docs.python.org/3/tutorial/