The 4 MCPs That Finish a Claude Code or Codex Build

How context.dev, Chrome DevTools MCP, Supabase MCP, and Vercel MCP take a build from a chat prompt to a live, tested app.

Why I added these four

Claude Code and Codex are good at writing code from a prompt. They are not good, on their own, at knowing whether an API changed last month, whether the page they just built actually renders, what is really sitting in your database, or how to get the finished app onto a real URL. Those four gaps are exactly what context.dev, Chrome DevTools MCP, Supabase MCP, and Vercel MCP close.

Without them, the AI works from training data and whatever you paste into the chat. With them, it can pull current documentation before it writes a line, open a real browser to test what it built, read and query your live database, and push the app live, all inside the same conversation you're already having with it.

The pattern before all four MCPs, if you've built with Claude Code or Codex for a while, is familiar: you ask for a feature, it writes something plausible, and you find out ten minutes later that the library's API changed, or the button doesn't actually fire, or the query it assumed would work doesn't match your real schema. None of that is the model being careless. It just has no way to check. These four MCPs are the checking.

01

How it actually works

Each MCP plugs into a different point in the same build loop: research, build, test, and ship. None of the four overlap, which is why running all of them together changes the whole workflow instead of just fixing one weak spot.

1. context.dev feeds the AI the latest official documentation for the library or API it's working with, so it writes against current syntax instead of outdated patterns it memorized during training.
2. Chrome DevTools MCP hands the AI control of a real browser. Once it writes a piece of UI, it can open the page, click through it, and check whether the result actually matches what it intended.
3. Supabase MCP and Vercel MCP close the last two gaps: Supabase MCP reads your database and runs queries directly, and Vercel MCP deploys the finished build, so the same conversation that started with a prompt ends with a live, tested app.

02

The one thing almost everyone gets wrong

People install all four MCPs and then keep working exactly like before, describing what they want and trusting the output without asking the AI to actually verify it.

> **WATCH OUT**
>
> Adding Chrome DevTools MCP doesn't automatically mean the AI tests its own work. Ask it directly: 'open the page in the browser and confirm the button fires' or 'query the table and confirm the row was written.' The MCP gives it the ability to check. You still have to ask it to check.

The same goes for Supabase MCP. Having database access doesn't mean the AI reads the schema before it writes a query against it, unless you tell it to. I build the checking step into the prompt itself now, not as an afterthought after something breaks. The four prompts further down are the exact phrasing I use to make that automatic.

03

The four MCPs, what each one replaces

Laid side by side, the pattern is the same for all four: each one turns a step you used to do by hand, in a separate tab, into something the AI can do inside the conversation.

MCP SERVER	WHAT IT GIVES THE AI	WHAT YOU'D DO INSTEAD, BY HAND
context.dev	Current official documentation for the API or library in play	Search the docs yourself and paste the relevant snippet into the chat
Chrome DevTools MCP	Control of a real, running browser	Open the app yourself after every change and click through it to check
Supabase MCP	Direct read and query access to your live database	Run the query in a separate tool and paste the result back into the chat
Vercel MCP	A deploy pipeline it can trigger on its own	Open the Vercel dashboard and deploy the build manually

---

Set it up in Claude Code or Codex

1. Both Claude Code and Codex add MCP servers through their own configuration, either a command that registers the server or an entry in the project's MCP settings file. The exact syntax is documented by each agent and can change, so use the agent's current MCP docs rather than an old command you find elsewhere.
2. For each of the four servers, go to that provider's own MCP setup page (context.dev, Chrome DevTools MCP, Supabase, and Vercel each publish one) and copy the exact connection command or URL they give you. Providers update these, so pulling it fresh avoids a stale command that silently fails to connect.
3. Some of these servers need an API key or a project reference (Supabase and Vercel both do, since they're pointing at your account). Ask Claude or Codex to add the key to your shell profile once you have it, then open a new terminal so the key is picked up.
4. Run a first test for each one once it's added: ask the AI to look up one piece of current documentation through context.dev, open a page through Chrome DevTools MCP, run one read-only query through Supabase MCP, and check the deploy status through Vercel MCP. If all four respond, the server list in your agent's tool output should show all four connected.
5. A successful first run looks like this: the AI answers with a real doc excerpt (not a guess), it can tell you what it saw when it opened the browser, it returns actual rows from your database, and it can tell you the current deploy status of your project without you opening any dashboard yourself.

> 💡 **QUICK TIPS**
>
> - Point Supabase MCP at a staging or development database first, not production, until you trust how the AI is using it.
> - If a server shows up in your tool list but every call fails, the API key is the most common cause. Re-check it before assuming the server itself is broken.
> - Add these one at a time instead of all four at once. If something breaks after adding a server, you'll know exactly which one to look at.

04

The commands I actually run

These are the four prompts I reach for most, one per MCP, written so the AI can't skip the verification step. Swap in your own feature, page, table, and condition.

> **CHECK THE DOCS BEFORE BUILDING**
>
> ```
> Before you write [FEATURE], use context.dev to pull the current documentation for [LIBRARY/API]. Confirm the method signatures you're about to use still match what's current, then build it.
> ```

> **TEST WHAT WAS JUST BUILT**
>
> ```
> Open [PAGE/ROUTE] in the browser using Chrome DevTools MCP. Click through [SPECIFIC ACTION] and tell me exactly what happened, including any console errors.
> ```

CHECK THE REAL DATA BEFORE ASSUMING

```
Use Supabase MCP to query [TABLE NAME] and show me the actual rows for [CONDITION].
Don't assume the schema, read it directly.
```

SHIP IT

```
Once [FEATURE] passes the browser test, use Vercel MCP to deploy the current build and
tell me the deploy status and the live URL when it's done.
```

When it breaks

Most of what goes wrong with these four traces back to one of two things: the agent hasn't picked up a newly added server yet, or a credential is missing or expired. Check those two first before you assume the server itself is broken.

ERROR	FIX
An MCP server shows in the config but the AI says it has no such tool	Restart the agent (Claude Code or Codex) after adding a new MCP server. New servers usually need a fresh session to load into the tool list.
context.dev or another MCP returns stale or generic answers	Check that the server actually connected and isn't silently falling back to the model's own training data. Ask the AI directly which source it pulled the answer from.
Chrome DevTools MCP can't open the page	Confirm the local dev server is actually running before asking the AI to open it. The MCP controls a browser, it doesn't start your app for you.
Supabase MCP queries fail with an auth error	The API key or project reference is usually wrong or expired. Re-add it to your shell profile and open a new terminal.
Vercel MCP deploy fails but a manual deploy from the dashboard works	Check whether the MCP is pointed at the right project and branch. A mismatched project reference is the most common cause.