Les 15 — Lesstof

MCP — Model Context Protocol

Vak: AI-Assisted Development **Opleiding:** NOVI Hogeschool Utrecht **Vorige les:** Les 14 — Cursor + Vercel **Volgende les:** Les 16 — RAG + Embeddings

Doel van deze les

Aan het einde van deze les heb je:

- Begrip van wat MCP is en welk probleem het oplost
- Vier MCPs gekoppeld in Cursor: **GitHub, Supabase, Vercel, Polderfest**
- De Polderfest MCP server lokaal gedraaid + via drie chats naar productie:
- **Chat 1:** GitHub MCP — repo aanmaken + remote toevoegen
- **Chat 2:** Polderfest + Supabase MCP — nieuwe feature bouwen
- **Chat 3:** Vercel MCP — project + deploy
- Productie-URL gekoppeld in `mcp.json`
- Optioneel: eigen feature toegevoegd via Cursor + Supabase MCP

1. Wat is MCP

Model Context Protocol = open standaard voor hoe AI-clients (Cursor, Claude Desktop) verbinding maken met externe tools en data.

Het probleem dat het oplost

Voor MCP had elke AI-client eigen plug-in/extension-systeem:

- Cursor: extension API
- Claude Desktop: plugins
- ChatGPT: GPTs + Custom Actions
- Windsurf, Zed: eigen API's

Eén feature × vier clients = vier integraties.

De oplossing

Eén protocol. Eén server. Vele clients. Anthropic introduceerde MCP in november 2024. Open spec.

Analogie: USB-C voor AI — één stekker, elke laptop.

2. Architectuur + primitives

Drie rollen:

Rol	Voorbeeld	Wat het doet
Host	Cursor, Claude Desktop	de app waarin je werkt
Client	MCP library	praat met server (zit in host)
Server	GitHub MCP, jouw Polderfest	biedt tools/resources/prompts

Drie primitives:

Primitive	Wat
Tools	functies (de AI doet iets)
Resources	read-only data (de AI leest iets)
Prompts	templates / persona's

Vandaag focus op **tools**.

3. Transport

In Cursor 3 is HTTP/SSE de moderne manier:

Transport	Wanneer
HTTP (remote)	Server is een webservice (Vercel, mcp.supabase.com, jouw eigen)
HTTP (lokaal)	npm run dev op localhost:3000
stdio	Oud — Cursor start lokaal proces. Vermijden waar mogelijk.

Alle MCPs van vandaag werken via HTTP — dat is wat we gaan zien.

4. Vier MCPs vandaag

MCP	URL	Auth
GitHub	https://api.githubcopilot.com/mcp/	Bearer (ghp_... PAT)
Supabase	https://mcp.supabase.com/mcp?project_ref=<dev_project>	Bearer (sbp_... PAT)
Vercel	https://mcp.vercel.com	OAuth (browser-flow)
Polderfest (eigen)	http://localhost:3000/api/mcp	geen

5. De `mcp.json` config

Plaats: `~/.cursor/mcp.json` (globaal — werkt overal) of `.cursor/mcp.json` in een project (alleen daar).

```
{
  "mcpServers": {
    "supabase": {
      "type": "http",
      "url": "https://mcp.supabase.com/mcp?project_ref=<JOUW_REF>&read_only=false",
      "headers": { "Authorization": "Bearer sbp_..." }
    },
    "github": {
      "type": "http",
      "url": "https://api.githubcopilot.com/mcp/",
      "headers": { "Authorization": "Bearer ghp_..." }
    },
    "vercel": {
      "type": "http",
      "url": "https://mcp.vercel.com"
    },
    "polderfest": {
      "type": "http",
      "url": "http://localhost:3000/api/mcp"
    }
  }
}
```

Patroon: `type: http` + `url` + optioneel `headers.Authorization` met Bearer.

Vercel heeft geen header — die doet OAuth bij eerste gebruik via een "Needs login" prompt in Cursor.

6. Tokens regelen

GitHub PAT (fine-grained)

1. github.com → **Settings** → **Developer settings** → **Personal access tokens** → **Fine-grained**
2. **Repository access:** kies welke repos je MCP mag zien (niet "All" als het niet hoeft)
3. **Permissions:** Contents (read+write), Issues (read+write), Pull requests (read+write)
4. Genereer → kopieer `ghp_...` (éénmalig zichtbaar)

Supabase Personal Access Token

1. supabase.com → **Account** → **Access Tokens** → **Generate new token**
2. Naam: bv. `cursor-mcp`
3. Kopieer `sbp_...` (éénmalig zichtbaar)

■■ **Dit is NIET** de project anon key of service role key. Dit is een **account-level** token — geeft toegang tot ALLE projecten in je account.

`project_ref` vind je in je Supabase project URL:
`https://supabase.com/dashboard/project/XXXXXX` — die XXXXXX is je ref.

Vercel OAuth

Geen token vooraf nodig. Bij eerste gebruik in Cursor verschijnt een "Needs login" knop in de MCP-indicator. Klik → autoriseer in browser → klaar.

7. De drie chats workflow

De climax van de les. Drie aparte Cursor-chats, elk met één focus.

Chat 1 — GitHub MCP: repo + remote

```
Maak een nieuwe public GitHub repo "polderfest-mcp" aan via GitHub MCP.
Voeg toe als remote `origin` aan deze folder.
Push de main branch.
```

GitHub MCP roept `create_repository` aan. Cursor draait daarna `git remote add` + `git push` in terminal. Geen `gh repo create` zelf typen.

Chat 2 — Polderfest + Supabase MCP: feature bouwen

```
Ik wil een review-systeem.

Stap 1: vraag Supabase MCP om een tabel band_reviews
  (id, band_id FK naar bands, score 1-10, comment, created_at)

Stap 2: voeg twee tools toe aan @file:app/api/mcp/route.ts:
- addReview(bandName, score, comment)
- getReviews(bandName)
```

Drie MCPs samen aan het werk:

- **Supabase MCP** maakt de tabel
- **Cursor** schrijft de tool-code
- **Polderfest MCP** voert de tools uit (na push + auto-deploy)

Chat 3 — Vercel MCP: project + deploy

```
Maak via Vercel MCP een nieuw project voor mijn GitHub repo polderfest-mcp.
Voeg env vars toe (production): SUPABASE_URL + SUPABASE_SERVICE_ROLE_KEY.
Deploy naar productie.
```

Vercel MCP doet: `create_project` → link GitHub → set env vars → deploy. Productie-URL terug.

Tot slot: vervang polderfest entry in `mcp.json` naar de productie-URL. Werkt vanaf dan zonder lokale dev-server.

8. Polderfest MCP — anatomie

`polderfest-mcp-finished.zip` is een complete Next.js-app:

```
polderfest-mcp/
■■■ app/
■   ■■■ api/mcp/route.ts   ← MCP server (het echte werk)
■   ■■■ lib/supabase.ts    ← Supabase client + capitalizeDay helper
■   ■■■ page.tsx           ← landingspagina
■   ■■■ layout.tsx
■■■ package.json           ← mcp-handler + @modelcontextprotocol/sdk + zod
■■■ .env.example
■■■ README.md
```

Vier tools

Tool	Wat
searchBands	zoek bands op query / genre / dag
getStageSchedule	schema voor 1 festivaldag
festivalStats	totalen + verdeling per genre / dag / stage
getWeather	Open-Meteo forecast voor festivaldag

Tool-anatomie

```
server.tool(
  "searchBands",
  "Zoek bands op query, genre, of dag.",   // ← description voor de AI
  {
    query: z.string().optional(),
    genre: z.string().optional(),
    day: z.enum(["vrijdag", "zaterdag", "zondag"]).optional(),
  },
  async ({ query, genre, day }) => {
    let q = supabase.from("bands").select("*").limit(50);
    if (query) q = q.ilike("name", `%${query}%`);
    if (genre) q = q.ilike("genre", `%${genre}%`);
    if (day) q = q.eq("day", capitalizeDay(day));
    const { data } = await q;
    return { content: [{ type: "text", text: JSON.stringify(data, null, 2) }] };
  },
);
```

Bekend uit Les 12/13: Zod schemas. **Nieuw:** `server.tool()` signature + return met `content: [...]`.

Bouwstenen

Package	Wat
@modelcontextprotocol/sdk	officiële MCP SDK van Anthropic
mcp-handler	Vercel-adapter voor Next.js
zod	input schemas

9. Lokaal draaien

```
unzip polderfest-mcp-finished.zip
cd polderfest-mcp-finished
npm install
cp .env.example .env.local
# vul SUPABASE_URL + SUPABASE_SERVICE_ROLE_KEY (uit Les 11/12)
npm run dev
```

- Landingspagina: `http://localhost:3000`
- MCP endpoint: `http://localhost:3000/api/mcp`

Polderfest entry zit al in je `mcp.json` (lokaal). Herstart Cursor → 4 indicators groen.

10. Bekende MCP servers

Inspiratie:

- **GitHub** — issues, PRs, repos, commits, code-search
- **Supabase** — DB queries, schema, migrations
- **Vercel** — projects, deployments, env vars
- **Filesystem** — lees/schrijf folders
- **Slack** — kanaal-search, posts
- **Linear / Jira** — tickets vanuit IDE
- **Notion** — pages + databases
- **Brave Search** — web search
- **PostgreSQL** — direct SQL

Zoek op `cursor.com/mcp` of `github.com/modelcontextprotocol/servers`.

11. Veiligheid

- Tokens niet committen — `~/.cursor/mcp.json` staat al buiten projecten
- Supabase: `read_only=true` voor demo's met productie-data
- GitHub PAT: fine-grained met minimale repo-rechten
- Vercel: OAuth + autoriseer alleen vertrouwde IDE

12. Verder lezen

- **MCP spec** — `modelcontextprotocol.io`
- **Cursor MCP docs** — `cursor.com/docs/mcp`
- **Supabase MCP** — `supabase.com/docs/guides/getting-started/mcp`
- **Vercel MCP** — `vercel.com/docs/agent-resources/vercel-mcp`
- **GitHub MCP** — `github.com/github/github-mcp-server`

Samenvatting

Vandaag heb je:

- MCP gezien als de open standaard voor AI-clients
- Vier MCPs gekoppeld via HTTP-config in `~/.cursor/mcp.json`
- De drie-chats workflow doorlopen: GitHub repo → Polderfest feature → Vercel deploy
- Je eigen Polderfest MCP live gekregen op Vercel
- Drie MCPs samen zien werken in één chat (Cursor + Supabase + Polderfest)

Volgende les: RAG + Embeddings — AI antwoorden op basis van eigen documenten.