

Les 12 — Tool Calling

Laat AI zelf kiezen welke functie aan te roepen — Polderfest vervolg

Vak:	AI-Assisted Development
Opleiding:	NOVI Hogeschool Utrecht
Klas:	Klas A (3 uur fysiek, demo-driven)
Vervolg op:	Les 11 — Vercel AI SDK + Polderfest 2027

Leerdoelen

- Het schaalprobleem van context-all begrijpen + Tool Calling als oplossing
- Anatomie van een tool kennen: description + parameters + execute
- Tools definiëren met Zod schemas (incl. enums)
- stopWhen voor multi-step workflows inzetten
- Tool-invocations zichtbaar maken in de UI
- Edge cases en error handling correct implementeren
- Verschil read- vs write-tools en wanneer welke gebruiken

Inhoud

1. Het schaalprobleem dat we oplossen
2. Wat is Tool Calling?
3. Anatomie van een tool
4. Multi-step met stopWhen
5. Refactor: chat-route met tools
6. Tool-invocations in de UI
7. Edge cases & error handling
8. Tool Calling vs context-all — vergelijking
9. Best practices
10. Wat komt hierna? Agents teaser
11. Bronnen

1 Het schaalprobleem

In Les 11 stuurden we **alle 500 bands** mee als context bij elke request:

```
const { data: bands } = await supabase.from("bands").select("*");
const context = bands.map(b => `- ${b.name}...`).join("\n");
const system = `Hier zijn alle bands:\n${context}\n...`;
```

Werkt voor 500. Werkt niet voor 50.000.

Records	Tokens	Cost (gpt-4o-mini)	Probleem
500	~30.000	\$0.005	OK
5.000	~300.000	\$0.05	Te traag, context grenst aan limit
50.000	~3 miljoen	n.v.t.	Past niet in context

Daarnaast:

- Context is een **snapshot** — niet up-to-date
- **Geen write-acties** mogelijk
- AI moet zelf zoeken in lap tekst — inefficiënt

Tool Calling lost dit op.

2

Wat is Tool Calling?

In plaats van alle data mee te sturen, geef je AI **tools** (functies). AI ziet de vraag, kiest welke tool relevant is, roept 'm aan met de juiste parameters, en gebruikt het resultaat om te antwoorden.

Flow

```
User: "Welke bands op vrijdag op de Main Stage?"
↓
AI: kiest searchBands({ day: "Vrijdag", stage: "Main Stage" })
↓
Supabase: 12 bands terug
↓
AI: formuleert antwoord op basis van resultaat
```

Wat win je

- **Schaalbaar** — werkt voor 100 of 10 miljoen records
- **Real-time** — tool draait elke keer opnieuw, geen snapshot
- **Type-safe** — parameters via Zod, gevalideerd
- **Multi-step** — meerdere tools combineren
- **Write-acties** — AI kan ook iets in DB zetten

3

Anatomie van een tool

Drie verplichte delen.

```
import { tool } from "ai";
import { z } from "zod";

const searchBands = tool({
  description:
    "Zoek bands op dag, stage, genre, of tier.",
  inputSchema: z.object({
    day: z.enum(["Vrijdag", "Zaterdag", "Zondag"]).optional(),
    stage: z.string().optional(),
    genre: z.string().optional(),
    tier: z.enum(["headliner", "mid", "opener"]).optional(),
  }),
  execute: async ({ day, stage, genre, tier }) => {
    let q = supabase.from("bands").select("*");
    if (day) q = q.eq("day", day);
    /* ... */
    const { data, error } = await q.limit(20);
    if (error) return { error: error.message };
    return data;
  },
});
```

description

Wat doet de tool en wanneer gebruik je 'm? AI **kiest tools op basis hiervan**.

- Vaag: 'Doe iets met bands' — AI kiest verkeerd
- Goed: 'Zoek bands op dag, stage, genre of tier. Gebruik voor filtervragen.'

parameters

Zod schema. Type-safe. AI mag alleen geldige waarden invullen.

- `z.string()` — vrije tekst
- `z.enum([...])` — vaste keuze
- `z.number().min(1).max(100)` — getal met bounds
- `.optional()` — parameter mag weg
- `.describe('...')` — extra context voor AI

execute

Async functie. **Belangrijk:** errors als data terug, niet als exception:

```
// ■ Niet doen – AI ziet de error niet
execute: async (...) => {
  const data = await supabase.from(...).select();
  if (data.error) throw new Error(data.error.message);
}

// ■ Wel doen – AI kan dit zelf afhandelen
execute: async (...) => {
  const { data, error } = await supabase.from(...).select();
  if (error) return { error: error.message };
  return data;
}
```

4

Multi-step met stopWhen

Met `stopWhen: stepCountIs(5)` geef je AI toestemming om tot 5 keer een tool aan te roepen voordat hij definitief antwoordt.

```
const result = streamText({
  model: openai("gpt-4o-mini"),
  tools: { searchBands, getStats, getBandByName },
  stopWhen: stepCountIs(5),
  messages,
});
```

Voorbeeld

User: 'Vergelijk de top headliner met de drukst geplande opener.'

```
Stap 1: AI roept searchBands({ tier: "headliner" }) aan
→ 50 bands terug, top 1 gevonden
Stap 2: AI roept searchBands({ tier: "opener" }) aan
→ 100 bands terug, top 1 gevonden
Stap 3: AI vergelijkt + antwoordt
```

Wanneer welk step-limit?

Use case	stepCountIs(N)
Simpele query ('welke bands op vrijdag?')	1-2
Vergelijking ('X vs Y')	3-5
Onderzoek ('vat alle X samen')	5-10
Agentic ('plan mijn weekend') — volgende les	15-30+

Default: 1 stap. Expliciet zetten voor multi-step.

5 Refactor: chat-route met tools

De volledige refactored route na deze les:

```
import { streamText, tool, stepCountIs, convertToModelMessages } from "ai";
import { openai } from "@ai-sdk/openai";
import { createClient } from "@supabase/supabase-js";
import { z } from "zod";

const supabase = createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
);

const searchBands = tool({ /* zie boven */ });
const getStats = tool({ /* ... */ });
const getBandByName = tool({ /* ... */ });

export async function POST(req: Request) {
  const { messages } = await req.json();

  const system = `Je bent een festival-assistent.
  Gebruik de beschikbare tools om vragen te beantwoorden.
  Verzin nooit data. Antwoord in het Nederlands.`;

  const result = streamText({
    model: openai("gpt-4o-mini"),
    system,
    messages: convertToModelMessages(messages),
    tools: { searchBands, getStats, getBandByName },
    stopWhen: stepCountIs(5),
  });
  return result.toUIMessageStreamResponse();
}
```

Wat is veranderd t.o.v. Les 11

- Geen `.select("*")` op start
- Geen grote context-string in system prompt
- Tools-object toegevoegd
- `stopWhen: stepCountIs(5)` voor multi-step
- System prompt veel korter — alleen rol + regels

6

Tool-invocations in de UI

`useChat` retournt messages met **parts** — array van delen (tekst-parts én tool-invocation-parts).

```
{messages.map((m) => (
  <div key={m.id}>
    <strong>{m.role}</strong>
    {m.parts?.map((part, i) => {
      if (part.type === "text") {
        return <div key={i}>{part.text}</div>;
      }
      // v6: tool-parts heten `tool-<toolName>`
      if (part.type?.startsWith("tool-")) {
        const toolName = part.type.replace("tool-", "");
        return (
          <div key={i} className="bg-yellow-50 p-2 rounded">
            ■ {toolName} ({JSON.stringify(part.input)})
            {part.state === "output-available" && (
              <details>
                <summary>Toon resultaat</summary>
                <pre>{JSON.stringify(
                  part.output, null, 2
                )}</pre>
              </details>
            )}
          </div>
        );
      }
    })
    </div>
  )
  return null;
})}
</div>
))}
```

Waarom tool-invocations tonen?

- **Debug** — zien wat AI aanroept met welke args
- **Vertrouwen** — gebruiker ziet 'hij heeft echt iets opgezocht'
- **Demo** — voor presentaties, hackathons, onboarding

Part states (v6)

state	Wat
"input-streaming"	Tool args worden nog gestreamed
"input-available"	Args compleet, tool draait
"output-available"	Tool is gerund, resultaat beschikbaar
"output-error"	Tool gaf een error

7

Edge cases & error handling

Ongeldige input (enum-restrictie)

Vraag: 'Welke bands op Donderdag?'. AI ziet enum is [Vrijdag, Zaterdag, Zondag]. Donderdag past niet. AI weigert tool en legt uit.

Les: gebruik enums voor restricted values.

Lege resultaten

Tool returnt count 0. AI legt uit: 'Geen X gevonden.'

Les: lege array is OK, AI handelt af.

Database errors

```
execute: async (...) => {  
  const { data, error } = await supabase.from(...).select();  
  if (error) return { error: error.message };  
  return data;  
}
```

AI ziet { error: "... " }, communiceert dit netjes. **Niet** exception throwen — dan crasht de chat.

Write-tools en user-intent

Write-tools veranderen echt iets. Bescherm via:

1. **Descriptions:** 'Alleen gebruiken als gebruiker expliciet vraagt'
2. **Confirmation UI:** extra bevestiging vóór insert
3. **Permissions:** check ingelogd via auth.uid()

Demo: open. Productie: streng.

8

Tool Calling vs context-all

Aspect	Les 11 (context-all)	Les 12 (Tool Calling)
Tokens per call	~30.000 (500 bands)	~2.000 (tools + result)
Schaal	Tot ~1000 records	Tot duizenden
Live data	Snapshot bij chat-start	Actueel per call
Write operaties	Niet mogelijk	Wel (addFavorite)
Multi-step	Beperkt — reasoning	Native (stopWhen)
Cost	Hoger	Lager
Complexiteit	Lager	Iets hoger

Wanneer toch context-all?

- Hele kleine dataset (<100 records)
- Snel prototype
- Geen schaal nodig

Voor productie: bijna altijd Tool Calling.

Descriptions schrijven

- ■ 'zoek dingen' — te vaag
- ■ 'zoek bands' — nog steeds vaag
- ■ 'Zoek bands in line-up. Filter op dag, stage, genre of tier. Gebruik voor filtervragen.'

Parameter design

- Gebruik `enum` voor categorische waarden
- `.optional()` voor filter-parameters
- `.describe()` op elke parameter
- Houd parameter-sets klein (3-5 max)

Returns

- Retournt **JSON-serializable** waarden (geen `Date` direct)
- Errors als `{ error: '...' }` — niet throwen
- Beperk grootte van responses (`limit 20`)

System prompts

```
const system = `Je bent een festival-assistent.  
Gebruik de beschikbare tools om vragen te beantwoorden.  
  
Tips:  
- Voor "welke bands op X?" → searchBands  
- Voor "hoeveel" → getStats  
- Voor specifieke band → getBandByName  
  
Verzin nooit data. Bij errors leg netjes uit.`;
```

10 Wat komt hierna? Agents teaser

Volgende les: Agents (Les 13)

Tool Calling met `stopWhen: stepCountIs(5)` is de eerste stap richting agents. Volgende les: `stepCountIs(20)` en hoger, of zelfs custom stop-condities.

```
const result = streamText({
  model: openai("gpt-4o-mini"),
  tools: { ... },
  stopWhen: stepCountIs(30),
  experimental_telemetry: { isEnabled: true },
  messages,
});
```

Voorbeeld autonome workflow

User: 'Plan mijn volledige Polderfest weekend op basis van mijn smaak.'

1. Agent vraagt user smaakprofiel
2. `searchBands` per dag + genre
3. Filter op overlap-vermijding
4. `addFavorite` per geselecteerde band
5. `listFavorites` om finaal schema te tonen
6. Wijst conflicten aan in tijdsloten
7. Optimaliseert en herhaalt

30+ tool-calls in één user-request.

Daarna in deze leerlijn

- **Les 14:** RAG + embeddings — semantic search op tekst-corpora
- **Les 15-16:** Testing + Deployment + Performance
- **Les 17-18:** Eindopdracht-werkdagen + Pitch

11 Bronnen

Vercel AI SDK

- Tools-documentatie: ai-sdk.dev/docs/foundations/tools
- Agents + stopWhen: ai-sdk.dev/docs/foundations/agents
- streamText reference: ai-sdk.dev/docs/reference/ai-sdk-core/stream-text
- useChat reference: ai-sdk.dev/docs/reference/ai-sdk-ui/use-chat

Underlying providers

- OpenAI Function Calling: platform.openai.com/docs/guides/function-calling
- Anthropic Tool Use: docs.anthropic.com/en/docs/build-with-claude/tool-use

Zod

- Docs: zod.dev
- Schema reference: zod.dev/?id=primitives

Supabase JS

- Query builder: supabase.com/docs/reference/javascript/select
- Filters: supabase.com/docs/reference/javascript/using-filters