NOVI Hogeschool — AI Leerlijn

# Les 7
# Van In-Memory naar Supabase

Database koppelen aan je Next.js app

| | |
|---|---|
| **Les** | 7 van 18 |
| **Duur** | 3 uur (09:00 - 12:00) |
| **Onderwerp** | Supabase: database koppelen aan Next.js |
| **Vereisten** | Werkend QuickPoll project uit Les 6 |

# Inhoudsopgave

# Deel 1: Poll-app afmaken

We maken het stemmen in onze QuickPoll app werkend. Dit duurt ~30 minuten.

## Stap 1.1 — votePoll() functie toevoegen

We missen een functie om stemmen te verwerken. Open **lib/data.ts** en voeg onderaan toe:

```
export function votePoll(id: string, optionIndex: number): Poll | undefined {
  const poll = polls.find((p) => p.id === id);
  if (!poll) return undefined;
  if (optionIndex < 0 || optionIndex >= poll.options.length) return undefined;
  poll.votes[optionIndex]++;
  return poll;
}
```

De functie zoekt de poll op id, checkt of de index geldig is, verhoogt de votes, en returnt de poll.

## Stap 1.2 — POST route werkend maken

Open **app/api/polls/[id]/route.ts** en vervang de inhoud:

```
import { NextResponse } from "next/server";
import { votePoll } from "@/lib/data";

interface RouteParams {
  params: Promise<{ id: string }>;
}

export async function POST(request: Request, { params }: RouteParams) {
  const { id } = await params;
  const body = await request.json();
  const optionIndex = body.optionIndex;

  const updatedPoll = votePoll(id, optionIndex);

  if (!updatedPoll) {
    return NextResponse.json(
      { error: "Poll niet gevonden of ongeldige optie" },
      { status: 400 }
    );
  }

  return NextResponse.json(updatedPoll);
}
```

## Stap 1.3 — Server Component + VoteForm split

Dit is het belangrijkste patroon in Next.js: de **pagina** is een Server Component die data ophaalt. Alleen het interactieve stuk (stemmen) wordt een apart **Client Component**.

### Stap 1.3a — Maak components/VoteForm.tsx (Client Component):

```
'use client'

import { Poll } from "@/types";
import { PollItem } from "./PollItem";
import { useState } from "react";

export function VoteForm({ poll: initialPoll }: { poll: Poll }) {
  const [poll, setPoll] = useState(initialPoll);

  const onVote = async (option: string) => {
    const optionIndex = poll.options.indexOf(option);
    const response = await fetch(`/api/polls/${poll.id}`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ optionIndex }),
    });
    if (response.ok) { setPoll(await response.json()); }
  };

  return <PollItem poll={poll} onOptionClick={onVote} />;
}
```

### Stap 1.3b — Open app/poll/[id]/page.tsx (Server Component):

```
import { getPollById } from "@/lib/data";
import { VoteForm } from "@/components/VoteForm";
import { notFound } from "next/navigation";

interface PageProps {
  params: Promise<{ id: string }>;
}

export default async function PollPage({ params }: PageProps) {
  const { id } = await params;
  const poll = getPollById(id);
  if (!poll) notFound();

  return (
    <div className="w-full p-4">
      <h2>{poll.question}</h2>
      <VoteForm poll={poll} />
    </div>
  );
}
```

**Het patroon:** Server Component haalt data op, Client Component doet interactie. Geen useEffect, geen loading state voor de initiële load. Dit patroon verandert niet als we straks

Supabase koppelen.

## Stap 1.4 — GET route toevoegen

We hebben een GET route nodig zodat de detail pagina de poll kan ophalen. Voeg toe in **app/api/polls/[id]/route.ts** (boven de POST):

```
import { getPollById, votePoll } from "@/lib/data";

export async function GET(request: Request, { params }: RouteParams) {
  const { id } = await params;
  const poll = getPollById(id);
  if (!poll) {
    return NextResponse.json({ error: "Poll niet gevonden" }, { status: 404 });
  }
  return NextResponse.json(poll);
}
```

## Stap 1.5 — Visuele feedback met percentage bars

Open **components/PollItem.tsx** en vervang de inhoud voor visuele stemresultaten:

```
'use client'
import { Poll } from "@/types"

type PollItemOptionProps = {
  option: string; votes: number; percentage: number;
  onClick?: (option: string) => void;
}

export const PollItemOption = ({ option, votes, percentage, onClick }:
PollItemOptionProps) => (
  <div onClick={() => onClick?.(option)}
    className="relative my-2 p-3 border rounded cursor-pointer hover:bg-gray-50">
    <div className="absolute top-0 left-0 h-full bg-blue-100 transition-all"
      style={{ width: `${percentage}%` }} />
    <div className="relative flex justify-between">
      <span>{option}</span>
      <span className="text-gray-500">{votes} ({percentage}%)</span>
    </div>
  </div>
)
```

**Check: start npm run dev, ga naar een poll, stem — je ziet de bars animeren!**

# Deel 2: Introductie Supabase — No Code

## Stap 2.1 — Wat is Supabase?

Supabase is een open-source Firebase alternatief. Het biedt:

**PostgreSQL database** — professionele SQL database (30+ jaar)
**Authenticatie** — login, registratie, OAuth (Google, GitHub)
**Storage** — bestanden uploaden
**Realtime** — live updates als data verandert
**Edge Functions** — serverless functies

Gratis tier beschikbaar voor leren en kleine projecten.

## Stap 2.2 — Supabase project aanmaken

1. Ga naar **supabase.com** en klik **Start your project**
2. Log in met **GitHub**
3. Klik **New Project**
4. Vul in: Project name: **quickpoll**, Region: **West EU (Frankfurt)**
5. Genereer een database password en sla het op
6. Klik **Create new project** en wacht ~30 seconden

## Stap 2.3 — polls tabel aanmaken

In de **Table Editor**, klik **Create a new table**:

| Kolom | Type | Opties |
|---|---|---|
| id | uuid | Primary key (auto) |
| created_at | timestamptz | Default: now() |
| question | text | Not null |

## Stap 2.4 — options tabel aanmaken

Maak een nieuwe tabel **options**:

| Kolom | Type | Opties |
|---|---|---|
| id | uuid | Primary key (auto) |
| created_at | timestamptz | Default: now() |
| poll_id | uuid | Foreign key → polls.id, CASCADE |
| text | text | Not null |
| votes | int8 | Default: 0 |

**Foreign key:** poll_id verwijst naar polls.id. **CASCADE** = als een poll wordt verwijderd, verdwijnen de opties ook.

## Stap 2.5 — RLS policies instellen

Row Level Security bepaalt wie je data mag lezen/schrijven. Maak deze policies:

| Tabel | Policy naam | Operatie | Rol | USING |
|---|---|---|---|---|
| polls | Allow public read | SELECT | anon | true |
| options | Allow public read | SELECT | anon | true |
| options | Allow public vote | UPDATE | anon | true |

## Stap 2.6 — Testdata toevoegen

Voeg via de Table Editor twee polls toe en 4 opties per poll (dezelfde als in de code). Kopieer de poll id's voor de options tabel!

## Stap 2.7 — SQL Editor

Probeer deze queries in de SQL Editor:

```
-- Alle polls ophalen
SELECT * FROM polls;

-- Opties van een specifieke poll
SELECT * FROM options WHERE poll_id = 'jouw-poll-id';

-- JOIN: polls met hun opties
SELECT polls.question, options.text, options.votes
FROM polls
JOIN options ON options.poll_id = polls.id;
```

# Deel 3: Supabase koppelen aan Next.js

## Stap 3.1 — Installatie

```
npm install @supabase/supabase-js
```

## Stap 3.2 — Environment variables

Ga naar **Settings** → **API** in Supabase. Kopieer de URL en anon key. Maak **.env.local**:

```
NEXT_PUBLIC_SUPABASE_URL=https://jouw-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJhbG...
```

NEXT_PUBLIC_ prefix = ook beschikbaar in de browser. De anon key is veilig om public te gebruiken — de beveiliging zit in de RLS policies.

## Stap 3.3 — Supabase client

Maak **lib/supabase.ts**:

```
import { createClient } from "@supabase/supabase-js";

const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL!;
const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!;

export const supabase = createClient(supabaseUrl, supabaseAnonKey);
```

## Stap 3.4 — Types updaten

Open **types/index.ts** en vervang:

```
export interface Poll {
  id: string;
  question: string;
  created_at: string;
  options: Option[];
}

export interface Option {
  id: string;
  poll_id: string;
  text: string;
  votes: number;
}
```

## Stap 3.5 — data.ts herschrijven voor Supabase

Vervang de hele inhoud van **lib/data.ts**:

```
import { supabase } from "./supabase";
import { Poll, Option } from "@/types";

export async function getPolls(): Promise<Poll[]> {
  const { data: polls, error } = await supabase
    .from("polls")
    .select("*, options(*)")
    .order("created_at", { ascending: false });
  if (error) { console.error(error); return []; }
  return polls || [];
}

export async function getPollById(id: string): Promise<Poll | null> {
  const { data: poll, error } = await supabase
    .from("polls")
    .select("*, options(*)")
    .eq("id", id)
    .single();
  if (error) { console.error(error); return null; }
  return poll;
}

export async function votePoll(pollId: string, optionId: string): Promise<Option |
null> {
  const { data: option } = await supabase
    .from("options").select("votes").eq("id", optionId).single();
  if (!option) return null;

  const { data: updated, error } = await supabase
    .from("options")
    .update({ votes: option.votes + 1 })
    .eq("id", optionId)
    .select().single();
  if (error) { console.error(error); return null; }
  return updated;
}
```

**select("*, options(*)")** haalt automatisch de gerelateerde opties op via de foreign key.

## Stap 3.6 — Homepage aanpassen

Open **app/page.tsx**:

```
import { getPolls } from "@/lib/data";
import { PollItem } from "@/components/PollItem";
import Link from "next/link";

export default async function Home() {
  const polls = await getPolls();
  return (
```

```
    <div className="w-full p-4">
      <h2>Onze polls</h2>
      {polls.map((poll) => (
        <Link key={poll.id} href={`/poll/${poll.id}`}>
          <PollItem poll={poll} />
        </Link>
      ))}
    </div>
  );
}
```

De functie is nu **async** + **await**. Dit kan omdat het een Server Component is.

## Stap 3.7 — PollItem aanpassen

Update **components/PollItem.tsx** om met het Option type te werken:

```
'use client'
import { Poll, Option } from "@/types"

type PollItemProps = { poll: Poll; onOptionClick?: (option: Option) => void }

export const PollItemOption = ({ option, percentage, onClick }) => (
  <div onClick={() => onClick?.(option)}
    className="relative my-2 p-3 border rounded cursor-pointer">
    <div className="absolute top-0 left-0 h-full bg-blue-100"
      style={{ width: `${percentage}%` }} />
    <div className="relative flex justify-between">
      <span>{option.text}</span>
      <span>{option.votes} ({percentage}%)</span>
    </div>
  </div>
)

export const PollItem = ({ poll, onOptionClick }: PollItemProps) => {
  const totalVotes = poll.options.reduce((sum, opt) => sum + opt.votes, 0);
  return (
    <section className="w-full my-6">
      <h2>{poll.question}</h2>
      {poll.options.map((option) => {
        const pct = totalVotes > 0 ? Math.round((option.votes / totalVotes) * 100) : 0;
        return <PollItemOption key={option.id} option={option} percentage={pct}
onClick={onOptionClick} />
      })}
    </section>
  );
}
```

## Stap 3.8 — VoteForm + detail pagina aanpassen

Update **VoteForm.tsx** — stuur nu **optionId** (uuid) mee i.p.v. index:

```
const onVote = async (option: Option) => {
  const response = await fetch(`/api/polls/${poll.id}`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ optionId: option.id }),
  });
  if (response.ok) { setPoll(await response.json()); }
};
```

In **page.tsx**: het enige verschil is **await** voor getPollById (nu async). De structuur verandert niet!

## Stap 3.9 — API routes aanpassen

Beide functies zijn nu **async** en gebruiken **await**:

```
export async function GET(request: Request, { params }: RouteParams) {
  const { id } = await params;
  const poll = await getPollById(id);
  if (!poll) return NextResponse.json({ error: "Niet gevonden" }, { status: 404 });
  return NextResponse.json(poll);
}

export async function POST(request: Request, { params }: RouteParams) {
  const { id } = await params;
  const { optionId } = await request.json();
  await votePoll(id, optionId);
  const poll = await getPollById(id);
  return NextResponse.json(poll);
}
```

**Check: herstart de dev server, stem, refresh — de stem is bewaard!**

# Huiswerk

## Opdracht: /create pagina bouwen

**1.** Maak een **/create** pagina met een formulier:
- Een invoerveld voor de vraag
- Minimaal 2 invoervelden voor opties
- Een "Voeg optie toe" knop
- Een submit knop

**2.** Maak de bijbehorende logica:
- INSERT de poll in Supabase (polls tabel)
- INSERT de opties in Supabase (options tabel, met poll_id)
- Redirect naar de homepage na aanmaken

**3.** Voeg een **"Nieuwe Poll"** link toe in de navbar

**4.** Test: maak een poll aan, stem erop, check in Supabase Table Editor

## Extra uitdaging

- Validatie: minimaal 2 opties, vraag mag niet leeg zijn
- Error handling: toon een melding als het aanmaken mislukt
- SQL queries schrijven in de Supabase SQL Editor:
  - Welke poll heeft de meeste stemmen?
  - Sorteer alle opties op votes (hoog naar laag)
- Push je code naar GitHub

### Inleveren voor de volgende les!