Les 5: QuickPoll

Bouw een poll-applicatie met Next.js

Wat ga je bouwen?
Een complete poll-applicatie waarmee gebruikers polls kunnen bekijken, stemmen, en nieuwe polls aanmaken. Je gebruikt: App Router, Server & Client Components, API Routes, Middleware, TypeScript, en Tailwind CSS.

Onderdeel	Details
Tijd	~75 minuten (les) + thuis afmaken
Werkwijze	Individueel
Starter	les5-quickpoll-starter.zip (download van Teams)
Tech Stack	Next.js 15, TypeScript, Tailwind CSS
Cursor	Begin ZONDER Cursor Tab. Pas later aanzetten.

Stappenplan overzicht:

Stap	Wat	Gegeven?
0	Setup: npm install, npm run dev	Compleet
1	Navigatiebalk bouwen in layout.tsx	Scaffold
2	Homepage: polls tonen	Scaffold
3	API Route: GET /api/polls/[id]	Scaffold
4	API Route: POST /api/polls/[id]/vote	Scaffold
5	Poll Detail pagina	Scaffold
6	VoteForm Client Component	Scaffold + hints
7	Loading, Error & Not Found	Scaffold

Bonus	Create Poll pagina	Alleen hints

STAP 0 — Setup

De starter zip bevat een compleet Next.js project met alle bestanden al aangemaakt. Je hoeft alleen **npm install** te draaien en je kunt beginnen.

```
# Unzip het project
unzip les5-quickpoll-starter.zip
cd quickpoll-starter

# Installeer dependencies
npm install

# Start de development server
npm run dev
```

Open **http://localhost:3000** in je browser. Je ziet de basis app.

Wat zit er al in de starter?

De volgende bestanden zijn **compleet** en hoef je niet aan te passen:

Bestand	Wat het doet
`src/types/index.ts`	Poll en CreatePollBody TypeScript interfaces
`src/lib/data.ts`	In-memory data met 3 polls + helper functies
`src/middleware.ts`	Request logging voor /api/* en /poll/* routes
`src/app/api/polls/route.ts`	GET alle polls + POST nieuwe poll
`.cursorrules`	Cursor configuratie voor Next.js

Bekijk eerst de complete bestanden!
Open `src/types/index.ts` en `src/lib/data.ts` en lees de code. Je moet begrijpen welke functies beschikbaar zijn (getPolls, getPollById, votePoll, createPoll) voordat je begint.

Bekijk ook `src/app/api/polls/route.ts` — dit is een **compleet voorbeeld** van een API route. Je gaat dit patroon straks zelf toepassen in stap 3 en 4.

STAP 1 — Navigatiebalk bouwen

`src/app/layout.tsx`

De layout.tsx is de root wrapper van je hele app. Alles wat je hier zet (navigatie, footer) verschijnt op **elke pagina**. De `{children}` prop is de pagina-inhoud die wisselt.

Wat moet je doen?

Vervang het commentaar in layout.tsx met een navigatiebalk:

- Een <nav> element met een witte achtergrond en border-bottom
- Links: logo/titel "QuickPoll" die linkt naar `/`
- Rechts: link naar `/` ("Polls") en `/create` ("Nieuwe Poll")

Belangrijk: gebruik <Link> van `next/link` in plaats van <a> tags. Link is al geimporteerd bovenaan het bestand.

Handige Tailwind classes:

```
// Navigatie container
"bg-white border-b border-gray-200 shadow-sm"

// Inner container (gecentreerd, max breedte)
"max-w-4xl mx-auto px-4 py-4 flex items-center justify-between"

// Logo stijl
"text-xl font-bold text-purple-600"

// Button stijl voor "Nieuwe Poll"
"bg-purple-600 text-white px-4 py-2 rounded-lg hover:bg-purple-700"
```

Check: Refresh je browser. Je zou een navigatiebalk moeten zien met links.

STAP 2 — Homepage: polls tonen

`src/app/page.tsx`

De homepage is een **Server Component** (geen "use client" nodig). Je kunt direct de `getPolls()` functie aanroepen — geen fetch, geen useEffect, geen loading state.

Wat moet je doen?

1. Roep `getPolls()` aan (al geimporteerd)
2. Maak een helper functie voor totaal stemmen:

```
const totalVotes = (poll: Poll): number =>
  poll.votes.reduce((sum, v) => sum + v, 0);
```

3. Map over de polls en toon voor elk:
   - De vraag (`poll.question`)
   - Aantal opties + aantal stemmen
   - De opties als kleine tags/badges
   - Alles gewrapped in een `<Link href={`/poll/${poll.id}`}>`

JSX structuur hint:

```
{polls.map((poll) => (
  <Link key={poll.id} href={`/poll/${poll.id}`}
    className="block bg-white rounded-xl border p-6 hover:border-purple-300"
  >
    <h2 className="text-lg font-semibold">{poll.question}</h2>
    <div className="text-sm text-gray-500">
      {poll.options.length} opties · {totalVotes(poll)} stemmen
    </div>
    {/* Toon hier de opties als badges */}
  </Link>
))}
```

Check: Je homepage toont nu 3 polls met hun opties. Klikken doet nog niks (poll detail pagina is nog leeg).

STAP 3 — API Route: GET enkele poll

`src/app/api/polls/[id]/route.ts`

Je gaat het patroon van `api/polls/route.ts` (GET all polls) toepassen op een **dynamische** API route. De `[id]` in de foldernaam wordt een parameter.

Kijk eerst naar het voorbeeld!
Open `src/app/api/polls/route.ts` en bestudeer hoe de GET functie werkt. Je gaat hetzelfde patroon volgen, maar dan met een parameter.

Wat moet je doen?

1. Haal het `id` op uit params (let op: het is een Promise!):

```
const { id } = await params;
```

2. Zoek de poll met `getPollById(id)`

3. Als de poll niet bestaat, return een 404:

```
return NextResponse.json(
  { error: "Poll niet gevonden" },
  { status: 404 }
);
```

4. Als de poll wel bestaat, return hem als JSON:

```
return NextResponse.json(poll);
```

Check: Ga naar `http://localhost:3000/api/polls/1` in je browser. Je zou JSON moeten zien met de eerste poll.

STAP 4 — API Route: POST stem

`src/app/api/polls/[id]/vote/route.ts`

Deze route handelt stemmen af. De client stuurt een POST met `{ optionIndex: number }` en de API update de stemmen in de data.

Wat moet je doen?

1. Haal `id` uit params
2. Lees de request body:

```
const body: VoteBody = await request.json();
```

3. Valideer: is `body.optionIndex` een number?

```
if (typeof body.optionIndex !== "number") {
  return NextResponse.json(
    { error: "optionIndex is verplicht" },
    { status: 400 }
  );
}
```

4. Roep `votePoll(id, body.optionIndex)` aan
5. Als het resultaat `undefined` is: return 404
6. Anders: return de geupdate poll als JSON

Testen? API routes met POST kun je niet in de browser testen. Gebruik je terminal:

```
curl -X POST http://localhost:3000/api/polls/1/vote -H "Content-Type: application/json" -d '{"optionIndex": 0}'
```

STAP 5 — Poll Detail pagina

`src/app/poll/[id]/page.tsx`

De poll detail pagina is een **Server Component** die de poll ophaalt en de VoteForm (Client Component) rendert. Metadata is al geregeld.

Wat moet je doen?

Vervang de placeholder in de PollPage functie:

1. Haal `id` op uit params
2. Zoek de poll met `getPollById(id)`
3. Als de poll niet bestaat: `notFound()` aanroepen
4. Render de vraag en de VoteForm:

```
return (
  <div className="max-w-2xl mx-auto">
    <h1 className="text-2xl font-bold text-gray-900 mb-6">
      {poll.question}
    </h1>
    <VoteForm poll={poll} />
  </div>
);
```

Let op: `notFound()` is al geimporteerd van `next/navigation`. Als je het aanroept, toont Next.js automatisch `not-found.tsx`.

Check: Ga naar `http://localhost:3000/poll/1`. Je ziet de vraag en de placeholder tekst van VoteForm.

STAP 6 — VoteForm: de stem interface

`src/components/VoteForm.tsx`

Dit is het meest uitdagende onderdeel. De VoteForm is een **Client Component** met state management en een API call. De state variabelen en helper functies zijn al voor je klaargezet.

Deel A: Opties tonen + selecteren

Map over `currentPoll.options` en maak klikbare knoppen:

```
{currentPoll.options.map((option, index) => (
  <button
    key={index}
    onClick={() => !hasVoted && setSelectedOption(index)}
    disabled={hasVoted}
    className={`w-full text-left p-4 rounded-lg border-2 ${
      selectedOption === index
        ? "border-purple-500 bg-purple-50"
        : "border-gray-200 hover:border-purple-300"
    }`}
  >
    <span className="font-medium">{option}</span>
  </button>
))}
```

Deel B: Stem knop + API call

Voeg onder de opties een "Stem!" knop toe die `handleVote()` aanroept. Vul de `handleVote` functie in:

```
const response = await fetch(`/api/polls/${currentPoll.id}/vote`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ optionIndex: selectedOption }),
});
if (response.ok) {
  const updatedPoll: Poll = await response.json();
  setCurrentPoll(updatedPoll);
  setHasVoted(true);
}
```

Deel C: Resultaten tonen (na het stemmen)

Als `hasVoted === true`, toon dan voor elke optie het percentage. Gebruik de `getPercentage()` functie die al in het bestand staat. Dit deel is aan jou — experimenteer met Tailwind voor een mooie weergave!

Check: Je kunt een optie selecteren, op "Stem!" klikken, en de resultaten worden getoond. Refresh de pagina en je kunt opnieuw stemmen.

STAP 7 — Loading, Error & Not Found

De bestanden staan al klaar met basis implementaties. Verbeter ze met betere styling:

loading.tsx — Skeleton loader

Vervang "Laden..." met een skeleton die lijkt op de echte content. Gebruik `animate-pulse` met grijze blokken:

```
<div className="animate-pulse space-y-4">
  <div className="h-8 bg-gray-200 rounded w-1/3" />
  <div className="h-4 bg-gray-200 rounded w-1/2 mb-8" />
  {[1, 2, 3].map((i) => (
    <div key={i} className="bg-white rounded-xl border p-6">
      <div className="h-5 bg-gray-200 rounded w-3/4 mb-3" />
      <div className="h-4 bg-gray-200 rounded w-1/4" />
    </div>
  ))}
</div>
```

error.tsx — Error boundary

Style de error pagina met Tailwind. De props (`error` en `reset`) zijn al getypt.

not-found.tsx — 404 pagina

De basis staat er al. Voeg styling toe naar wens.

BONUS — Create Poll pagina

`src/app/create/page.tsx`

Bouw een formulier om nieuwe polls aan te maken. Dit is een Client Component. Je hebt nodig:

- `useState` voor question, options (string[]), en isSubmitting
- Een input voor de vraag
- Dynamische inputs voor opties (min 2, max 6)
- POST naar `/api/polls` met de juiste body
- Na success: `router.push("/")`

HULP NODIG?

Veelvoorkomende foutmeldingen:

Foutmelding	Oorzaak	Oplossing
"use client" must be at the top of the file	useState in een Server Component	Voeg "use client" toe bovenaan het bestand
params is not iterable or similar	params niet ge-await	const { id } = await params;
Cannot find module "@/types"	Import pad klopt niet	Check tsconfig.json paths of herstart dev server
404 op API route	route.ts op verkeerde plek	Check folder structuur: app/api/polls/[id]/route.ts
Type error: Property does not exist	TypeScript type mismatch	Check je interface in types/index.ts

Handige commando's:

```
# Dev server starten
npm run dev

# TypeScript check (zonder te builden)
npx tsc --noEmit

# API route testen met curl
curl http://localhost:3000/api/polls
curl http://localhost:3000/api/polls/1
curl -X POST http://localhost:3000/api/polls/1/vote \
  -H "Content-Type: application/json" \
  -d '{"optionIndex": 0}'
```

Kom je er niet uit?

1. Kijk naar de slides (cheat sheet op slide 26!)
2. Vraag je buurman/buurvrouw
3. Vraag Tim

Je hoeft niet alles in de les af te hebben. Maak thuis af wat je niet af kreeg.