

Les 5 — Lesstof

Next.js Basics — het React framework

Vak: Technical Foundations
Cursus: AI Developer
Volgende les: Les 6 — QuickPoll Part 2

Vak: Technical Foundations **Vorige les:** Les 4 — TypeScript Fundamentals **Volgende les:** Les 6 — Next.js QuickPoll deel 2

Inhoud

1. [Waarom Next.js](#1-waarom-nextjs)
2. [App Router vs Pages Router](#2-app-router-vs-pages-router)
3. [Setup — eerste project](#3-setup--eerste-project)
4. [Folder-structuur](#4-folder-structuur)
5. [Pages + Layouts](#5-pages--layouts)
6. [Server Components vs Client Components](#6-server-components-vs-client-components)
7. [Routing — dynamic + nested](#7-routing--dynamic--nested)
8. [Data fetching basics](#8-data-fetching-basics)
9. [Styling met Tailwind](#9-styling-met-tailwind)
10. [Loading + Error states](#10-loading--error-states)
11. [QuickPoll — het project van deze les](#11-quickpoll--het-project-van-deze-les)

1. Waarom Next.js

React alleen is een UI-library — niet meer. Voor een echte website heb je nodig:

- Routing (welke URL → welke pagina)
- Server-side rendering (SEO + snelle first paint)
- API endpoints
- Image optimization, fonts, caching

- Deployment

Next.js voegt dit allemaal toe bovenop React. Standaard framework van Vercel.

Waarom Next.js voor deze leerlijn

- Industrie-standaard sinds 2022
- Werkt naadloos met Vercel deploy (Les 15)
- Server Components + Client Components — modern paradigm
- TypeScript first-class
- AI tools (Cursor, v0) zijn geoptimaliseerd voor Next.js

Andere keuzes (Remix, Astro, SvelteKit) bestaan, maar Next.js is de breedst gebruikte.

2. App Router vs Pages Router

Next.js heeft twee router-systemen — historisch gegroeid:

Pages Router (legacy, sinds 2016)

- `pages/` folder
- File = route. `pages/about.tsx` → `/about`
- `_app.tsx`, `_document.tsx`, `getServerSideProps`, ...

App Router (sinds Next.js 13, default sinds 14)

- `app/` folder
- File = route. `app/about/page.tsx` → `/about`
- Server Components by default
- Layouts, nested routing, loading states als file
- React Server Components paradigm

In deze leerlijn: App Router. Alle nieuwe projecten = App Router.

3. Setup — eerste project

Create

```
pnpm create next-app@latest quickpoll \
  --typescript --tailwind --app --no-src-dir --import-alias "@/*"
cd quickpoll
pnpm dev
```

Open `localhost:3000` → werkt.

Wat krijg je out-of-the-box

- TypeScript geconfigureerd
- Tailwind CSS v4
- ESLint
- Hot reloading
- File-based routing
- Optimized fonts (Geist)

4. Folder-structuur

```
quickpoll/
├── app/
│   ├── layout.tsx      # Root layout – wraps every page
│   ├── page.tsx        # Homepage (/)
│   ├── globals.css     # Tailwind imports
│   ├── about/
│   │   └── page.tsx    # /about
│   └── poll/
│       ├── [id]/
│       │   └── page.tsx # /poll/123 (dynamic)
│       └── components/ # Herbruikbare components
│           ├── lib/    # Utility code
│           └── public/  # Static files
├── next.config.ts
├── tsconfig.json
└── package.json
```

File-conventies

Filename	Wat
page.tsx	De pagina-component voor deze route
layout.tsx	Wraps children met persistent UI
loading.tsx	Loading state
error.tsx	Error boundary
not-found.tsx	404 voor deze route
route.ts	API endpoint

5. Pages + Layouts

Page

```
// app/about/page.tsx
export default function AboutPage() {
  return <h1>About us</h1>;
}
```

Layout

```
// app/layout.tsx
export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="nl">
      <body>
        <nav>...</nav>
        {children}
        <footer>...</footer>
      </body>
    </html>
  );
}
```

Layouts zijn **persistent** — niet ge-remount tussen pagina-navigaties. State blijft.

Nested layouts

```
app/
├── layout.tsx           # Root – alle pagina's
├── (marketing)/
│   ├── layout.tsx      # Layout voor marketing pages
│   ├── page.tsx
│   └── about/page.tsx
└── dashboard/
    ├── layout.tsx      # Layout voor dashboard
    └── page.tsx
```

(marketing) = "route group" — geen URL-segment, alleen voor file-organisatie.

6. Server Components vs Client Components

App Router's grootste paradigm-shift.

Server Component (default)

```
// app/page.tsx – runt op de server
export default async function Home() {
  const data = await fetch("https://api.example.com/posts").then(r => r.json());
  return <ul>{data.map(p => <li key={p.id}>{p.title}</li>)}</ul>;
}
```

- Draait alleen op server
- Geen useState, geen useEffect, geen browser-APIs
- Mag async zijn — await data fetching direct
- Geen JavaScript naar client (kleinere bundle)

Client Component

```
// components/Counter.tsx
"use client";
import { useState } from "react";

export function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>{count}</button>;
}
```

- "use client" directive bovenaan
- Gewoon React — hooks, events, state
- Wordt JavaScript naar browser

Wanneer welke

	Server	Client
Data fetching	✓	(kan, niet ideaal)
Hooks (useState, etc.)	✗	✓
Event handlers (onClick)	✗	✓
Browser APIs (localStorage)	✗	✓
SEO content	✓	(werkt, minder ideaal)

Vuistregel: Server by default. "use client" alleen als nodig.

7. Routing — dynamic + nested

Static route

```
app/about/page.tsx → /about
```

Dynamic route

```
app/poll/[id]/page.tsx → /poll/abc-123
```

In de component:

```
export default async function Poll({ params }: { params: Promise<{ id: string }> }) {  
  const { id } = await params;  
  return <div>Poll {id}</div>;  
}
```

Vanaf Next.js 16: params is een Promise (was object).

Catch-all routes

```
app/docs/[...slug]/page.tsx → /docs/a/b/c
```

`params.slug = ["a", "b", "c"]`.

Link tussen pages

```
import Link from "next/link";  
  
<Link href="/about">About</Link>  
<Link href={` /poll/${id} `}>Bekijk poll</Link>
```

Link = client-side navigatie (geen full page reload).

8. Data fetching basics

In Server Component

```

async function getPolls() {
  const res = await fetch("https://api.example.com/polls", {
    next: { revalidate: 60 }, // cache 60 sec
  });
  return res.json();
}

export default async function Home() {
  const polls = await getPolls();
  return <PollList polls={polls} />;
}

```

Cache opties

```

fetch(url, { cache: "force-cache" }) // permanent (default)
fetch(url, { next: { revalidate: 60 } }) // ISR – herval na 60s
fetch(url, { cache: "no-store" }) // nooit cachen (live)

```

In Client Component

```

"use client";
import { useEffect, useState } from "react";

export function Live() {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch("/api/data").then(r => r.json()).then(setData);
  }, []);
  return data ? <p>{data.value}</p> : <p>Loading...</p>;
}

```

Voor interactiviteit. Of via TanStack Query (komt later).

9. Styling met Tailwind

Tailwind = utility-first CSS framework. Class per CSS-property.

Basics

```

<div className="flex items-center gap-4 p-4 bg-white rounded-lg shadow">
  <h2 className="text-2xl font-bold text-gray-900">Hello</h2>
  <p className="text-sm text-gray-500">World</p>
</div>

```

Responsive

```
<div className="text-sm md:text-base lg:text-lg">
```

- Default = mobile
- md: = tablet+ (768px+)
- lg: = desktop+ (1024px+)

Hover, focus, dark

```
<button className="bg-blue-500 hover:bg-blue-600 focus:ring-2 dark:bg-blue-700">
```

Wanneer custom CSS

99% niet nodig. Bij echt aparte animations, complexe selectors → CSS module.

10. Loading + Error states

loading.tsx

```
// app/poll/[id]/loading.tsx
export default function Loading() {
  return <div className="animate-pulse">Loading poll...</div>;
}
```

Toont automatisch wanneer page-component nog await is.

error.tsx

```
"use client";
export default function Error({ error, reset }: {
  error: Error;
  reset: () => void;
}) {
  return (
    <div>
      <h2>Oops: {error.message}</h2>
      <button onClick={reset}>Probeer opnieuw</button>
    </div>
  );
}
```

Vangt errors in deze route subtree.

not-found.tsx

```
export default function NotFound() {
  return <h1>404 – Pagina niet gevonden</h1>;
}
```

Custom 404 voor deze route. Trigger met `notFound()` helper.

11. QuickPoll — het project van deze les

We bouwen **QuickPoll** — een mini-app voor snelle peilingen. Stack: Next.js + TypeScript + Tailwind.

Wat we deze les bouwen (Part 1)

- Setup van het project
- Homepage met form om nieuwe poll te maken
- Hardcoded poll-data in een TS-file (in-memory)
- Detail-pagina voor een poll
- Voting werkt via button-click (lokaal in state)

Volgende les (Les 6 — Part 2)

- Server actions voor vote submission
- Loading + error states
- Polish van de UI
- Setup voor database (komt in Les 7)

Architectuur

```
app/
  page.tsx           # Homepage – lijst polls + create form
  poll/
    [id]/
      page.tsx       # Detail page met voting
  components/
    PollForm.tsx     # Client – create poll form
    VoteForm.tsx     # Client – stemmen
  lib/
    data.ts          # In-memory poll store
```

In Les 7 vervangen we `lib/data.ts` door echte Supabase queries — en QuickPoll wordt productie-ready.

Bronnen

- **Next.js docs:** <https://nextjs.org/docs>

-
- **App Router intro:** <https://nextjs.org/docs/app/getting-started>
 - **Server Components:** <https://nextjs.org/docs/app/getting-started/server-and-client-components>
 - **Tailwind v4 docs:** <https://tailwindcss.com/docs>
 - **Next.js Learn (interactief):** <https://nextjs.org/learn>
 - **shadcn/ui (component library):** <https://ui.shadcn.com>